Fundamentos de programación en Python

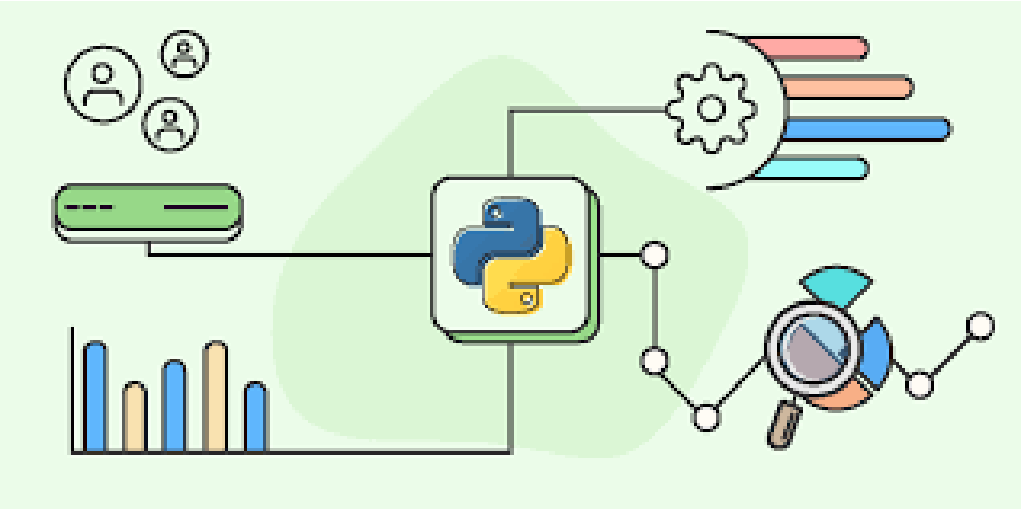
Manejo de excepciones

Especialización en Ciencia de Datos

2021

Objetivos

- Aprender sobre Excepciones
- Codificar un algoritmo manejando excepciones para control de errores.



Contenido:

1. Control de Excepciones
2. Excepciones pre-construidas
3. Levantar Excepciones
4. Excepciones personalizadas

1. Control de Excepciones

Excepciones

A veces, cuando incluso una pieza de código o rutina puede estar correcta, hay condiciones en las cuales algo puede ir por mal camino. Por ejemplo, cuando se pide en una lista un elemento cuyo índice está fuera del rango de índices de dicha lasita. O bien, en alguna operación en donde se intenta hacer una división y ésta es por cero. O bien, cuando se intenta abrir un archivo y éste se encuentra bloqueado o corrputo.

Las excepciones son errores que ocurren en tiempo de ejecución. Puede que no sean excepciones o errores letales, pero podría detener la ejecución de tu programa. Puede que a veces ese sea el comportamiento apropiado, pero en otros casos tal vez la mejor alternativa sea prevenirlo y controlarlo.

Ejemplos de Excepciones

Algunos ejemplos de excepciones.

```
# division por cero
divide = lambda x,y : x/y

divide(5,0)
```

```
---------------------------------------------------------------
---------------
ZeroDivisionError                         Traceback (most re
cent call last)
<ipython-input-2-f21df4fe471f> in <module>
      2 divide = lambda x,y : x/y
      3
----> 4 divide(5,0)

<ipython-input-2-f21df4fe471f> in <lambda>(x, y)
      1 # division por cero
----> 2 divide = lambda x,y : x/y
      3
      4 divide(5,0)

ZeroDivisionError: division by zero
```

```
lista = [i*3 for i in range(10)]
print(lista)

# indice fuera de rango
lista[11]
```

```
[0, 3, 6, 9, 12, 15, 18, 21, 24, 27]

---------------------------------------------------------------
-----------------
IndexError                                Traceback (most
recent call last)
<ipython-input-7-70cd0f3c8d5f> in <module>
      3
      4 # indice fuera de rango
----> 5 lista[11]

IndexError: list index out of range
```

Controlando una excepción

Para controlar una excepción, Python provee la sentencia **try**, el cual captura cualquier excepción que pudiera aparecer dándonos la posibilidad de programar un comportamiento frente a ese escenario.

Bloque de ejecución normal

Bloque de ejecución en caso que se lance una excepción

```
# sea la siguiente funcion
def divide(numerador, denominador):
    try:
        return numerador/denominador
    except:
        return 999999
```

```
print(divide(100,20))
print(divide(1000,0))
```

```
5.0
999999
```

Controlando una excepción específica

En este caso se ha controlado solamente la excepción ZeroDivisionError.

Bloque de ejecución normal

Bloque de ejecución en caso que se lance una excepción ZeroDivisionError

```
# sea la siguiente funcion
def divide(numerador, denominador):
    try:
        return numerador/denominador
    except ZeroDivisionError as e:
        return 999999
```

```
print(divide(100,20))
print(divide(1000,0))
```

```
5.0
999999
```

Controlando múltiples excepciones

Pongámonos en un caso más complejo, en donde existe la posibilidad que varias excepciones pudieran aparecer.

Esta función no controla excepciones, le haremos algunas modificaciones a continuación

```
# sea la siguiente funcion
def divide_listas(lista1,lista2):
    return [lista1[i]/lista2[i] for i in range(len(lista1))]
```

```
print(divide_listas([1,2,3],[4,5,6]))
```

```
[0.25, 0.4, 0.5]
```

Bloque de ejecución normal

Bloque de ejecución en caso que se lance una excepción IndexError

Bloque de ejecución en caso que se lance una excepción ZeroDivisionError

```
def divide_listas(lista1,lista2):
    try:
        return [lista1[i]/lista2[i] for i in range(len(lista1))]
    except IndexError as e1:
        print('Manejando error:',e1)
        return []
    except ZeroDivisionError as e2:
        print('Manejando error:',e2)
        return []
```

```
print(divide_listas([1,2,3,4],[4,5,6]))
```

```
Manejando error: list index out of range
[]
```

```
print(divide_listas([1,2,3],[4,0,6]))
```

```
Manejando error: division by zero
[]
```

Controlando múltiples excepciones

También es posible capturar una excepción en un bloque y todas las demás en otro bloque (bloque else).

En este bloque se capturas todas las excepciones que no se capturaron en los bloques anteriores.

```
def divide_listas(lista1,lista2):
    try:
        return [lista1[i]/lista2[i] for i in range(l
    except IndexError as e1:
        print('Manejando error:',e1)
        return []
    except ZeroDivisionError as e2:
        print('Manejando error:',e2)
        return []
    else:
        print('Unknown error')
```

```
print(divide_listas([1,2,3,4],[4,5,6]))
```

```
Manejando error: list index out of range
[]
```

Bloque de finalización

La sentencia try no solamente ofrece la posibilidad de dar control sobre las excepciones que pudieran ser lanzadas en la ejecución del código sino que también da la posibilidad de agregar un bloque de finalización a todo evento, es decir, que se ejecuta ya sea se haya realizado una ejecución normal o bien haya ocurrido una excepción. Recordemos la presentación de lectura de archivos, en donde utilizamos justamente el bloque try/finally para ejecutar un bloque de código a todo evento

```
try:
    fobj = open('es_olvido.txt', 'rt', encoding='utf-8')
    for line in fobj.readlines():
        print(line.strip())
finally:
    fobj.close()
```

```
Hoy es un día azul de primavera,
Creo que moriré de poesía,
De esa famosa joven melancólica
No recuerdo ni el nombre que tenía.
Sólo sé que pasó por este mundo
Como una paloma fugitiva:
La olvidé sin quererlo, lentamente,
Como todas las cosas de la vida.
```

Acá mejoramos el código e incorporamos un control de la excepción que se lanza cuando el archivo no es encontrado

```
try:
    fobj = open('no-existo.txt','rt')
    for linw in fobject.readlines():
        print(line.strip())
except FileNotFoundError as e1:
    print(e1)
finally:
    print('Bloque de finalización')
```

```
[Errno 2] No such file or directory: 'no-existo.txt'
Bloque de finalización
```

2. Excepciones pre-construidas

Excepciones preconstruidas

En Python, todas las excepciones deben ser instancias de una clase que derive de **BaseException**. Estas excepciones podrían ser utilizadas al momento de programar un código para que sean lanzadas o bien podrían extenderse en una subclase. A continuación, la jerarquía de excepciones de Python.

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      |    +-- ModuleNotFoundError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      |    +-- RecursionError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
      +-- SystemError
      +-- TypeError
      +-- ValueError
      |    +-- UnicodeError
      |         +-- UnicodeDecodeError
      |         +-- UnicodeEncodeError
      |         +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
           +-- EncodingWarning
           +-- ResourceWarning
```

https://docs.python.org/3/library/exceptions.html#exception-hierarchy

3. Levantando Excepciones

Levantando excepciones

Una pieza de código, sea una función, una método de una clase, o un programa que se está implementando, podría considerar como parte de su diseño el levantar una excepción bajo ciertas circunstancias. Para eso, hay que utilizar la sentencia **raise** seguido de la excepción que se desea propagar.

La función en este caso levanta una excepción en el caso que no se cumpla una condición de validación.

```
def asignar_nota(val):
    if 1 <= val <= 7:
        print(f'Nota {val} asignada')
    else:
        raise ValueError(f'Valor {val} invalido')
```

```
print(asignar_nota(5.5))
print(asignar_nota(8.5))
```

```
Nota 5.5 asignada
None

---------------------------------------------------
------------------------
ValueError                                Traceback
(most recent call last)
<ipython-input-43-a40c00e217bf> in <module>
      1 print(asignar_nota(5.5))
----> 2 print(asignar_nota(8.5))

<ipython-input-41-f237519eba5b> in asignar_nota(va
l)
      3         print(f'Nota {val} asignada')
      4     else:
----> 5         raise ValueError(f'Valor {val} inva
lido')

ValueError: Valor 8.5 invalido
```

4. Excepciones Personalizadas

Excepciones personalizadas

Como se pudo apreciar, Python tiene una serie de excepciones que pueden ser utilizadas en una pieza de código para ser lanzadas. Sin embargo, a veces puede ser necesario definir una excepción personalizada para dar solución a un caso de uso particular.

En ese caso, se puede hacer uso de los conocimientos de orientación a objetos y de herencia, de tal forma de definir una clase que herede de la clase **Exception** o de alguna de sus clases descendientes.

```
class SalaryNotInRange(Exception):
    def __init__(self, salary, message='''Salary
                                not in range'''):
        self._salary = salary
        self._message = message
        super().__init__(self._message)
```

```
salary = 1000
if not 5000 < salary < 15000:
    raise SalaryNotInRange(salary)
```

```
---------------------------------------------------
------------------------
SalaryNotInRange                          Traceback
(most recent call last)
<ipython-input-49-6dacfac19f79> in <module>
      1 salary = 1000
      2 if not 5000 < salary < 15000:
----> 3     raise SalaryNotInRange(salary)

SalaryNotInRange: Salary
                                not in range
```

KIBERNUM

Creando Juntos Excelencia

Software Engineering | IT Staffing | IT Academy | IT Consulting

Proyecto apoyado por CORFO