Estructura de Datos y POO

Semana 2

LA ANATOMÍA DE LA EXCEPCIÓN – MÉTODOS Y MANEJO DE CADENAS

Contenido

- Excepciones
- Cadenas

PYTHON

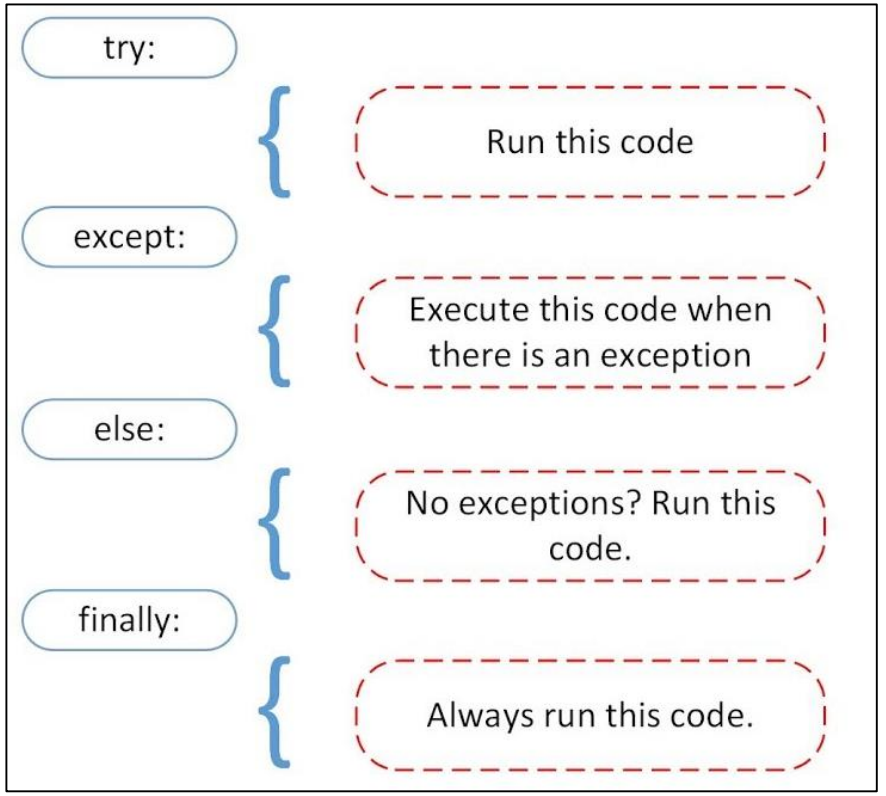
LA ANATOMÍA DE LA EXCEPCIÓN – MÉTODOS Y MANEJO DE CADENAS

Excepciones

Las excepciones son bloques de código que nos permiten continuar con la ejecución de un programa pese a que ocurra un error.
Cada vez que el código intenta hacer algo erróneo, irresponsable o inaplicable, **Python** hace dos cosas:

- Detiene el programa.
- Crea un tipo especial de dato, llamado excepción.

Ambas actividades llevan por nombre lanzar una excepción. Podemos decir que **Python** siempre lanza una excepción (o que una excepción ha sido lanzada) cuando no tiene idea de qué hacer con el código.



LA ANATOMÍA DE LA EXCEPCIÓN – MÉTODOS Y MANEJO DE CADENAS

Bloque try-except-else-finally

BLOQUES TRY – EXCEPT: Para prevenir el fallo debemos poner el código propenso a errores en un bloque **try** y luego encadenar un bloque **except** para tratar la situación excepcional mostrando que ha ocurrido un fallo. Esta forma nos permite controlar situaciones excepcionales que generalmente darían error y en su lugar mostrar un mensaje o ejecutar una pieza de código alternativo.

BLOQUE ELSE: Es posible encadenar un bloque **else** después del **except** para comprobar el caso en que todo funcione correctamente (no se ejecuta la excepción).

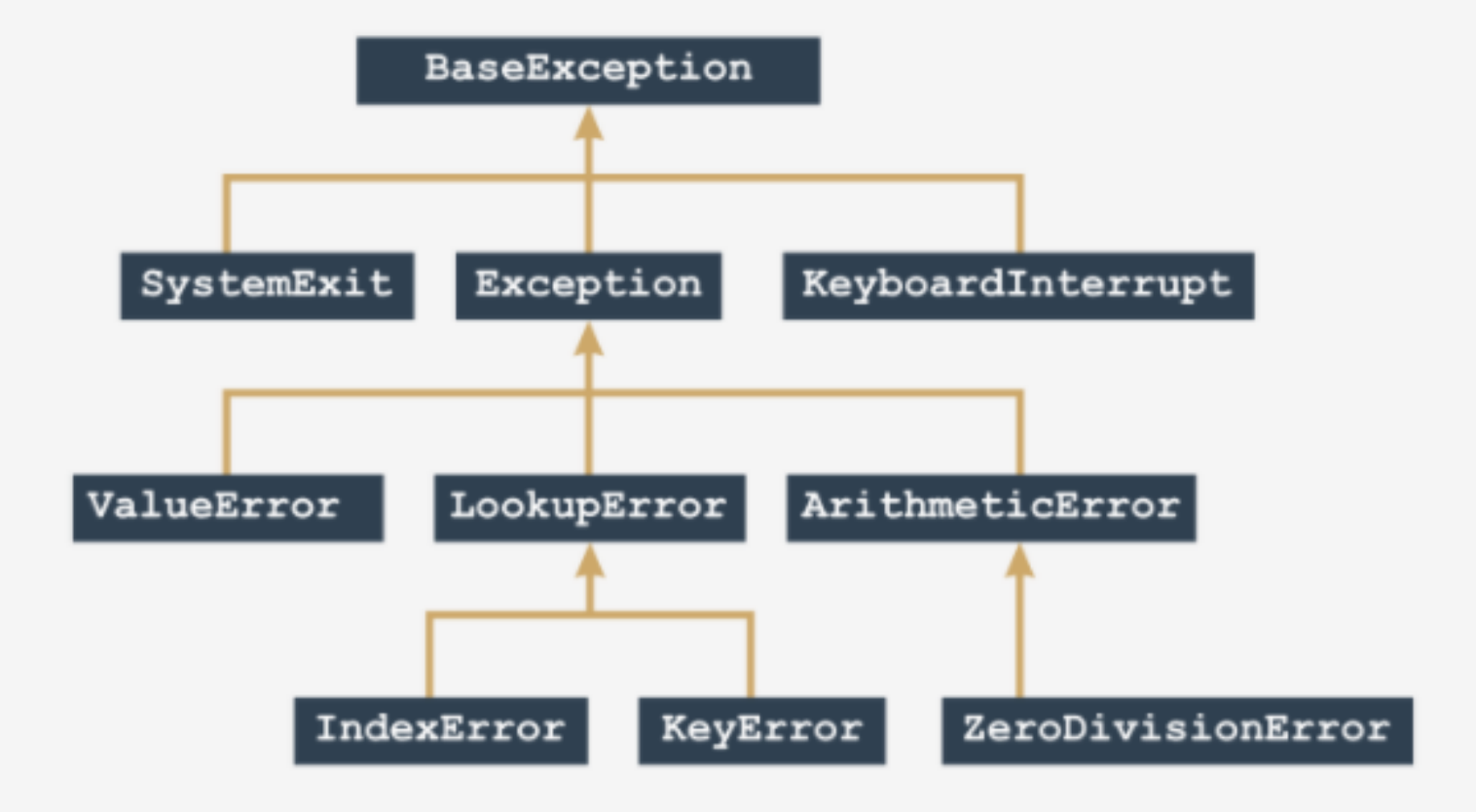
BLOQUE FINALLY: Por último, es posible utilizar un bloque **finally** que se ejecute al final del código, ocurra o no ocurra un error.

```
while(True):
    try:
        n = float(input("Introduce un número: "))
        m = 4
        print("{}/{} = {}".format(n,m,n/m))
    except:
        print("Ha ocurrido un error, introduce bien el número")
    else:
        print("Todo ha funcionado correctamente")
        break  # Importante romper la iteración si todo ha salido bien
    finally:
        print("Fin de la iteración") # Siempre se ejecuta
```

LA ANATOMÍA DE LA EXCEPCIÓN – MÉTODOS Y MANEJO DE CADENAS

Anatomía de la excepción

Python 3 define 63 excepciones integradas, algunas son más generales (incluyen otras excepciones) mientras que otras son completamente concretas (solo se representan a sí mismas). Mientras más arriba se encuentra una excepción, más general es. Por el contrario, las excepciones ubicadas en los extremos inferiores son más específicas.



LA ANATOMÍA DE LA EXCEPCIÓN – MÉTODOS Y MANEJO DE CADENAS

Cadenas

Una cadena es una **secuencia o sucesión inmutable** de caracteres (letras, números u otros signos o símbolos).

Las cadenas pueden ser delimitadas por comillas apostrofes o comillas (‘Estructura de Datos’ o “Estructura de Datos”). En el caso de cadenas multilínea, las cadenas se delimitan por tres apóstrofes o tres comillas.

```
1  linea = 'Linea #1'
2  print(len(linea))
3
4  linea = "Linea #2"
5  print(len(linea))
6
7  multiLinea = '''Linea #3
8  Linea #4'''
9  print(len(multiLinea))
10
11 multiLinea = """Linea #5
12 Linea #6"""
13 print(len(multiLinea))
```

LA ANATOMÍA DE LA EXCEPCIÓN – MÉTODOS Y MANEJO DE CADENAS

Indexación

Si desea acceder a cualquiera de los caracteres de una **cadena**, puede hacerlo utilizando la **indexación**. Tener en cuenta que si se intenta pasar los límites de la **cadena**, provocará una excepción.

Los **índices negativos** y las **rodajas** (rebanadas) tienen el mismo comportamiento que en una lista.

El operador **in** verifica si su argumento izquierdo (una cadena) se puede encontrar en cualquier lugar dentro del argumento derecho (otra cadena)

```
cadena = 'Estructura de Datos y Programación Orientada a Objetos'

print(cadena[0]) #E
print(cadena[-1]) #s
print(cadena[1:3]) #st
print(cadena[3:]) #ructura de Datos y Programación Orientada a Objetos
print(cadena[:3]) #Est
print(cadena[3:-2]) #ructura de Datos y Programación Orientada a Objet
print("P" in cadena) #True
print("p" in cadena) #False
print("ato" in cadena) #True
print(" y " in cadena) #True
print("OBJETOS" in cadena) #False
```

LA ANATOMÍA DE LA EXCEPCIÓN – MÉTODOS Y MANEJO DE CADENAS

Operaciones con cadenas

Concatenado
Operador: **+**
(necesita de 2 o más cadenas)
Replicado
Operador: ***
(necesita de un número y una cadena)
ord()
Obtiene el código ASCII de un carácter específico
chr()
Se tiene el código ASCII y se desea obtener el carácter correspondiente

```
c1 = "a"
c2 = "b"
print(c1 + c2) # ab
print(c2 + c1) #ba
print(5 * c1) #aaaaa
print(c2 * 4) #bbbb
print(ord(c1)) #97
print(ord(c2)) #98
print(chr(97)) #a
print(chr(98)) #b
```

LA ANATOMÍA DE LA EXCEPCIÓN – MÉTODOS Y MANEJO DE CADENAS

Operaciones con cadenas

min(): Encuentra el elemento mínimo de la cadena pasada como un argumento.

max(): Encuentra el elemento máximo de la cadena pasada como un argumento.

En ambos casos existe una condición - la cadena **no puede estar vacía**, de lo contrario se obtendrá una excepción **ValueError**

```
cadena = 'Estructura de Datos y Programación Orientada a Objetos'

print('[' + min(cadena) + ']') #[ ]
print('[' + max(cadena) + ']') #[ó]
```

LA ANATOMÍA DE LA EXCEPCIÓN – MÉTODOS Y MANEJO DE CADENAS

Operaciones con cadenas

index(): Devuelve el primer elemento del valor especificado en su argumento.
list(): Crea una lista que contiene todos los caracteres de la cadena (argumento).
count(): Cuenta todas las apariciones del elemento dentro de la cadena.
startswith()/endswith(): Comprueba si la cadena dada comienza/termina con el argumento especificado y devuelve True o False.
find()/rfind(): Busca una subcadena y devuelve el índice de la primera/última aparición, pero no genera un error si la subcadena no existe.

```
cadena = "Estructura de Datos y Programación Orientada a Objetos"
print(cadena.index("t")) #2
print(cadena.index("D")) #14

print(list(“Juan")) #[‘J', ‘u', ‘a', ‘n’]

print('Juan Francisco'.count("a")) #2

print(cadena.endswith("s")) #True
print(cadena.endswith("S")) #False
print(cadena.startswith(“E")) #True
print(cadena.startswith(“e")) #False
print(cadena.find("DXYZ")) #-1
print(cadena.find("tos")) #16
print(cadena.rfind("tos")) #51
```

LA ANATOMÍA DE LA EXCEPCIÓN – MÉTODOS Y MANEJO DE CADENAS

Operaciones con cadenas

isalnum(): Comprueba si la cadena contiene solo dígitos o caracteres alfabéticos (letras), devuelve True o False.

isalpha(): Comprueba si la cadena contiene solo caracteres alfabéticos (letras), devuelve True o False.

isdigit(): Comprueba si la cadena contiene solo dígitos, devuelve True o False.

```
print('Estructura'.isalnum()) #True
print('Qatar 2022'.isalnum()) #True
print('2022'.isalnum()) #True
print('Estructura de Datos'.isalnum()) #True
print('@'.isalnum()) #False
print('POO_2'.isalnum()) #False
print('Estructura'.isalpha()) #True
print('POO2'.isalpha()) #False
print('2021'.isalpha()) #False
print('Estructura'.isdigit()) #False
print('POO2'.isdigit()) #False
print('2021'.isdigit()) #True
```

LA ANATOMÍA DE LA EXCEPCIÓN – MÉTODOS Y MANEJO DE CADENAS

Operaciones con cadenas

islower(): Comprueba si la cadena contiene solo letras minúsculas, devuelve True o False.

isupper(): Comprueba si la cadena contiene solo letras mayúsculas, devuelve True o False.

isspace(): Comprueba si la cadena contiene solo espacios en blanco, devuelve True o False.

```
print("Juan".islower()) #False
print('juan'.islower()) #True

print("Juan".isupper()) #False
print('juan'.isupper()) #False
print('JUAN'.isupper()) #True

print(' \n '.isspace()) #True
print(" ".isspace()) #True
print("Juan Francisco".isspace()) #False
```

LA ANATOMÍA DE LA EXCEPCIÓN – MÉTODOS Y MANEJO DE CADENAS

Operaciones con cadenas

lower(): Hace una copia de la cadena original, reemplaza todas las letras mayúsculas con sus equivalentes en minúsculas.

upper(): Hace una copia de la cadena original, reemplaza todas las letras minúsculas con sus equivalentes en mayúsculas.

swapcase(): Crea una nueva cadena intercambiando todas las letras por mayúsculas o minúsculas dentro de la cadena original.

capitalize(): Crea una nueva cadena, si el primer carácter dentro de la cadena es una letra lo convertirá a mayúsculas.

title(): Cambia la primera letra de cada palabra a mayúsculas, convirtiendo todas las demás a minúsculas

```
print("JUAN".lower()) #juan
print("juan".upper()) #JUAN
print("juan francisco".capitalize())
#Juan francisco
print("Juan".swapcase()) #jUAN
print(“juan francisco".title())
#Juan Francisco
```

LA ANATOMÍA DE LA EXCEPCIÓN – MÉTODOS Y MANEJO DE CADENAS

Operaciones con cadenas

lstrip(): Sin parámetros devuelve una nueva cadena a partir de la original eliminando todos los espacios en blanco iniciales. Con un parámetro elimina también todos los caracteres incluidos en el argumento, no solo espacios en blanco.

rstrip(): Lo mismo que lstrip pero afecta el lado opuesto de la cadena.

strip(): Combina los efectos de lstrip y rstrip.

```
print(" IDAT".lstrip()) #IDAT
print("www.idat.edu.pe".lstrip("w.")) #idat.edu.pe
print("IDAT ".rstrip()) #IDAT
print("idat.com".lstrip(“.com")) #idat
print(“ juan ".strip()) #juan
print("xxjuanxx".strip("xx")) #juan
```

LA ANATOMÍA DE LA EXCEPCIÓN – MÉTODOS Y MANEJO DE CADENAS

Operaciones con cadenas

join(): Recibe una lista de cadenas, luego todos los elementos de la lista serán unidos en una sola cadena, la cadena desde la que se ha invocado el método será utilizada como separador.
split(): Divide la cadena y crea una lista de todas las subcadenas detectadas.
replace(): Con dos parámetros devuelve una copia de la cadena original en la que todas las apariciones del primer argumento han sido reemplazadas por el segundo argumento. Con tres parámetros emplea un tercer argumento para limitar el número de reemplazos.

```
print(",".join(["juan", "francisco", "fernandez"]))
#juan,francisco,fernandez

print("Juan;Francisco;Fernandez".split(";"))
['Juan', 'Francisco', 'Fernandez']

print("Juan Francisco Fernandez".replace("a", "A"))
#JuAn FrAncisco FernAndez
print("Juan Francisco Fernandez".replace("a", "A",2))
#JuAn FrAncisco Fernandez
```

LA ANATOMÍA DE LA EXCEPCIÓN – MÉTODOS Y MANEJO DE CADENAS

Comparación de cadenas

Las cadenas de Python se pueden comparar utilizando el mismo conjunto de operadores que se utilizan en relación con los números, compara valores de código ASCII, carácter por carácter.

```
print("Juan" == "Juan") #True
print("Juan" == "juan") #False
print("Juan" > "juan") #False
print("Juan Francisco" > "Juan") #True
```

Gracias