PYTHON II

Clase 2

FUNCIONES DE ORDEN SUPERIOR Y ANÓNIMAS

FUNCIONES DE ORDEN SUPERIOR

- Podemos crear funciones que pueden tomar funciones como parámetros y también retornar (devolver) funciones como resultado. Una función que hace ambas cosas, o alguna de ellas, se la llama función de orden superior. En el ejemplo genérico de la derecha se pasa una función y una lista para que la función superior() trabaje.
- Existen algunas funciones de orden superior incorporadas (built-in) en Python como map() y filter().

```
def sumar(x):
        return x+100
def cuadrado(x):
        return x**2
def superior(funcion,lista):
        resultado = [ ]
        for n in lista:
                resultado.append(funcion(n))
        return resultado
numeros = [2,5,10]
print(superior(sumar,numeros))
out: [102, 105, 110]
print(superior(cuadrado,numeros))
out: [4, 25, 100]
```

FUNCIONES ANÓNIMAS O LAMBDA

- Las funciones lambda son un tipo de funciones que típicamente se pueden escribir en una línea y cuyo código a ejecutar suele ser pequeño.
- Resulta complicado explicar las diferencias porque, básicamente, la diferencia es cómo se escribe, la sintaxis. Para que tengas una idea, son funciones que queremos usar y descartar. Idealmente se usan como argumento en funciones de orden superior.
- La función cuadrado(x) que usamos en el ejemplo anterior la podríamos escribir como:

```
lambda x : x**2
```

- Y la de sumar(x):

```
lambda x : x+100
```

Y con la función superior(funcion,lista) quedaría así:

```
print(superior(lambda x : x**2 , numeros))
```

out: [102, 105, 110]

```
print(superior(lambda x : x+100 , numeros))
```

out: [4, 25, 100]

- A las funciones anónimas también se les puede asignar un nombre como si fuera una variable (en realidad, como si fuera un objeto). Esto último no es muy aconsejable y no fueron pensadas para esto.
- Pero si estamos en la consola interactiva haciendo pruebas, puede ser de gran utilidad para definir pequeñas subrutinas.

```
>>> cuadrado = lambda x : x**2
>>> cuadrado(10)
100
```

EJERCICIO 1: ORDEN SUPERIOR

- Cree una función llamada “superior”, que reciba por argumento una función y una lista. La función que se pasa por argumento a superior debe elevar al cubo un número y retornarlo. Para que luego al aplicarla en la de orden superior, esa operación se realice miembro a miembro de la lista.
- Para finalizar la función “superior” debe de devolver una nueva lista

SOLUCIÓN – EJERCICIO 1

```
def superior(funcion,lista):
    nueva = []
    for n in lista:
        nueva.append(funcion(n))
    return nueva
```

```
##################################################
```

```
print(superior(lambda x : x**3,[1,2,3]))
```

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 “<stadin>”, 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 rang
```

- 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
```

LANZAR EXCEPCIONES

- Ahora volvamos al ejemplo de nuestra rudimentaria función sumar(), que se veía así:

```
def sumar(a, b):
    return a + b
```

Como se trata de una operación aritmética, lo ideal sería poder chequear que los argumentos sean números enteros (int) o de coma flotante (float). Si bien dos colecciones del mismo tipo pueden concatenarse vía el operador +, no es la intención de nuestra función.

Utilizando la función incorporada isinstance() podemos chequear si un objeto es de un tipo de dato determinado.

```
def sumar(a, b):
    if not isinstance(a, (int, float)) or not isinstance(b, (int, float)):
        raise TypeError("Se requieren dos numeros.")
    return a + b
print(sumar(7, 5)) #Imprime 12.
out: 12
print(sumar(2.5, 3.5)) # Imprime 6.
out: 6.0
print(sumar([1, 2], [3, 4])) # Lanza la excepción.
out:
raise TypeError("Se requieren dos numeros.")
 TypeError: Se requieren dos numerosnt, float)) ot not
```

- El segundo argumento de isinstance() puede ser un tipo de dato o una tupla. Si es un tipo de dato, retorna True si el primer argumento efectivamente coincide con ese tipo de dato; si es una tupla, retorna True si el primer argumento coincide con alguno de los tipos de datos contenidos en ella.
- Por convención, se estila lanzar TypeError cuando se quiere indicar que se obtuvo un argumento de un tipo inesperado, y ValueError cuando el tipo es correcto pero el valor no lo es.

EJERCICIO 2 - CALCULADORA

- Crear un programa que solicite dos números en consola e imprima el resultado de las cuatro operaciones aritméticas aplicadas sobre ellos. Por ejemplo (en rojo la entrada del usuario):

Escribe un número: 7

Escribe otro número: 5

a + b: 12

a - b: 2

a * b: 35

a / b: 1.4

Teniendo en cuenta lo siguiente:

- Si el usuario ingresa cualquier otra cosa que no sea un número, debe volver a preguntar, en ambos casos.
- Contemplar que el segundo número puede ser cero y, por ende, llegado el momento de la división el programa debe imprimir "No se puede dividir por cero".

Como restricción, no se pueden usar condicionales (if).

```
def solicitar_numero(mensaje):
    while True:
        try:
            return float(input(mensaje))
        except ValueError:
            print("Hay un error")


##############################################################


a = solicitar_numero("Escribe un número: ")
b = solicitar_numero("Escribe otro número: ")

print("a + b:", a + b)
print("a - b:", a - b)
print("a * b:", a * b)
try:
    print("a / b:", a / b)
except ZeroDivisionError:
    print("No se puede dividir por cero.")
```

EJERCICIO 3 -PAÍSES

- Crear un script que solicite al usuario el código de un país e imprima su nombre, de acuerdo con el siguiente diccionario:

```
# Diccionario código: país.
paises = {
        "ar": "Argentina",
        "es": "España",
        "us": "Estados Unidos",
        "fr": "Francia"
        }
```

Si el código ingresado no se encuentra en el diccionario, debe imprimir un mensaje en pantalla y volver a preguntar. Si el usuario escribe “salir”, el programa debe terminar

```
paises = {
    "ar": "Argentina",
    "es": "España",
    "us": "Estados Unidos",
    "fr": "Francia"
}

while True:
    codigo = input("Ingrese un codigo: ")
    if codigo == "salir":
        break
    try:
        print(paises[codigo])
    except KeyError:
        print("El código es invalido, vuelve a intentarlo.")
```