Funciones y Clases; Librerías

Funciones

Definir una funcion:

```
def nombre_funcion(parametro1, parametro2, ...):
    return valor_del_return
```

Llamar una funcion:

```
nombre_funcion(argumento1, argumento2, ...)
```

`return`: es opcional, pero sin return devuelve None
parametros por defecto: – siempre deben ser lo ultimo

`*args`: una tupla de argumentos sin limite
`**kwargs`: diccionarios cuyas keys se convierten en parámetros y sus valores en los argumentos de los parámetros

```
def nombre_funcion(parametros, *args, **kwargs,
                   parametro_por_defecto = valor)
    arg/kwarg: sin */** dentro de la funcion
    arg[0]
```

Llamar una funcion con *args:

```
nombre_funcion(argumento, argumento, argumento, ...)
```

o

```
nombre_funcion(*[lista_o_tupla_de_args])
```

Llamar una funcion con **kwargs:

```
nombre_funcion(**diccionario)
```

Clases

Definir una clase:

```
class NombreClase:

    def __init__(self, atributo1, atributo2):
        self.atributo1 = atributo1
        self.atributo2 = atributo2
        self.atributo_por_defecto = 'valor'

    def nombre_funcion1(self, parametros)
        self.atributo += 1
        return f"el nuevo valor es {self.atributo}"
```

Definir una clase hija:

```
class NombreClaseHija(NombreClaseMadre):
    def __init__(self, atributo1, atributo2):
    super().__init__(atributo_heredado1, ...)

    def nombre_funcion_hija (self, parametros):
```

Crear un objeto de la clase:

`variable_objeto = NombreClase(valor_atributo1, valor_atributo2)` instanciar (crear) un objeto
`variable_objeto.atributo` devuelve el valor del atributo guardado para ese objeto
`variable_objeto.atributo = nuevo_valor` para cambiar el valor del atributo
`variable_objeto.nombre_funcion()` llamar una funcion

`print(help(NombreClase)` imprime informacion sobre la clase

Regex

- una abreviatura de `expresión regular`, `regex` es una cadena de texto que permite crear patrones que ayudan a emparejar, localizar y gestionar strings

`import re` para poder trabajar con regex

Operadores communes de regex

`+` coincide con el carácter precedente una o más veces
`*` coincide con el carácter precedente cero o más veces u opcional
`?` indica cero o una ocurrencia del elemento precedente
`.` coincide con cualquier carácter individual
`^` coincide con la posición inicial de cualquier string
`$` coincide con la posición final de cualquier string

Sintaxis básica de regex

`\w` cualquier caracter de tipo alfabético
`\d` cualquier caracter de tipo númerico
`\s` espacios
`\n` saltos de línea
`\W` cualquier caracter que no sea una letra
`\D` cualquier caracter que no sea un dígitos
`\S` cualquier elemento que no sea un espacio
`()` aísla sólo una parte de nuestro patrón de búsqueda que queremos devolver
`[]` incluye todos los caracteres que queremos que coincidan e incluso incluye rangos como este: a-z y 0-9
`|` es como el operador 'or'
`\` señala una secuencia especial (escapar caracteres especiales)
`{}` Exactamente el número especificado de ocurrencias
`{n}` Exactamente n veces
`{n,}` Al menos n veces
`{n,m}` Entre n y m veces

Métodos Regex

`re.findall("patron", string)` busca en todo el string y devuelve una lista con todas las coincidencias en nuestro string

`re.search("patron", string_original)` busca en todo el string y devuelve un objeto con la primera coincidencia en nuestro string

`re.match("patron", "string_original)` busca en la primera linea del string y devuelve un objeto con la primera coincidencia en nuestro string

`resultado_match.span()` devuelve la referencia de las posiciones donde hizo el "match"

`resultado_match.group()` devuelve el element resultando de la coincidencia del "match"

`re.split("patron", "string_original")` busca en todo el string y devuelve una lista con los elementos separados por el patron

`re.sub("patron", "string_nuevo", "string_original")` busca en todo el string y devuelve un string con el element que coincide

Modulos/Librerias (paquetes de funciones)

Importar y usar modulos y sus funciones

`import modulo` para importar un modulo
`from modulo import funcion` importar solo una funcion
`modulo.funcion()` usar una funcion de un modulo
`modulo.clase.funcion()` para usar una funcion de una clase
`import modulo as md` asignar un alias a un modulo

Libreria os

`os.getcwd()` devuelve la ruta de donde estamos trabajando; se puede guardar en un variable e.g. ruta = os.getcwd()
`os.listdir()` devuelve una lista de los archivos y carpetas donde estamos trabajando
`os.listdir('carpeta')` devuelve los contenidos de otra carpeta
`os.chdir('ruta')` cambia la carpeta en la que estes
`os.mkdir('nueva_carpeta')` crear una nueva carpeta
`os.rename('nombre_carpeta', 'nueva_nombre')` cambia el nombre de una carpeta
`os.rmdir('carpeta')` borra la carpeta

Libreria shutil

from shutil inmport rmtree
`rmtree('carpeta')` borra la carpeta y subcarpetas

Abrir y cerrar ficheros

Primero hay que guardar la ruta del archivo:
ubicacion_carpeta = os.getcwd()
nombre_archivo = "text.txt"
`ubicacion_archivo = ubicacion_carpeta + "/" + nombre_archivo`

`f = open(ubicacion_archivo)` abrir un archivo en variable f
`f.close()` cerrar un archivo `* IMPORTANTE *`

```
with open(ubicacion_archivo) as f:
    codigo e.g. variable = f.read()
```

abre el archivo solo para ejecutar el codigo indicado (y despues lo deja)

Encoding

`from locale import getpreferredencoding`
`getpreferredencoding()` para saber que sistema de encoding estamos usando
`f = open(ubicacion_archivo, encoding="utf-8")` abrir un archivo y leerlo con el encoding usado; guardar con .read()

mode: argumento opcional al abrir un archivo

`r` – read
`w` – write - sobreescribe
`x` – exclusive creation, sólo crearlo si no existe todavía
`a` – appending, añadir texto al archivo sin manipular el texto que ya había
hay que anadir otra letra:
`t` – texto – leer en texto
`b` – bytes – leer en bytes (no se puede usar con encoding)

`f = open(ubicacion_archivo, mode = "rt")`

Leer ficheros

`f.read()` leer el contenido de un archivo
`f.read(n)` leer los primeros n caracteres de un archivo
`variable = f.read()` guardar el contenido del archivo (o n caracteres de un archivo) en un variable
`f.readline(n)` por defecto devuelve la primera linea o n lineas
`f.readlines()` devuelve una lista de todas las lineas del archivo (cada linea es un elemento); se usa vacio sin n y list_name[x:] para seleccionar lineas especificas

Escribir en ficheros

```
with open(ubicacion_archivo, "w") as f:
    f.write("Texto que va en el fichero.")
```

para escribir

```
with open(ubicacion_archivo, "a") as f:
    f.write("Texto que va en el fichero.")
```

para anadir texto
`f.writelines('lista')` para anadir lineas de texto de una lista

Ficheros xml

`import xml.etree.ElementTree as ET` importa la librería xml
`variable_tree = ET.parse('ruta/archivo.xml')` abre el archivo
`variable_root = variable_tree.getroot()` saca el elemento que envuelve todo (el elemento raíz) en una lista

```
<root>
    <child_tag atributo1="valor" atributo2=valor>
        <subchild_tag> elemento </subchild_tag>
    </child_tag>
</root>
```

`variable_root.tag` devuelve el nombre del tag del raiz
`variable_root.attrib` devuelve los atributos del fichero

`variable_root.find("tag").find("childtag").text` devuelve la primera ocasión en que el tag de un elemento coincida con el string
`variable_root.findall("tag").findall("childtag").text` devuelve todos los elementos cuyos tag coincide

MySQL Connector/Python

Conectar a una base de datos

`import mysql.connector` para importar MySQL Connector

pip install mysql-connector
pip install mysql-connector-Python

`connect()` para conectar a una base de datos:

```
variable_cnx = mysql.connector.connect(user='root',
                                       password='AlumnaAdalab',
                                       host='127.0.0.1',
                                       database='nombre_BBDD')
```

`from mysql.connector import errorcode` importar errores
`mysql.connector.Error` se puede usar en un try/except
`cnx.close()` desconectar de la base de datos

Realizar queries

`variable_cursor = cnx.cursor()` crear el objeto cursor que nos permite comunicar con la base de datos
`variable_cursor.close()` desconectar el cursor
`variable_query = ("SQL Query")` guardar un query en un variable
`variable_cursor.execute(variable_query)` ejecutar el query; devuelve una lista de tuplas
`import datetime` sacar fechas en el formato AAAA-MM-DD
`datetime.date(AAAA, M, D)` devuelve el formato de fecha
`variable_query = "SQL Query... %s AND %s")` query dinamica
`variable_cursor.execute(query, (variable1, variable2))` valores que van en lugar de los %s
`variable_cursor.execute("SHOW DATABASES")` mostrar las BBDD
`variable_cursor.execute("SHOW TABLES")` mostrar las tablas de la BBDD indicado en la conexión
`variable_cursor.execute("SHOW TABLES")`
`variable_cursor.execute("SHOW COLUMNS FROM bbdd.table")` mostrar las columnas de la tabla especificada; hay que conectarse a la bbdd information_schema

Argumentos cursor:

`variable_cursor = cnx.cursor([arg=value[, arg=value]...])`
`buffered=True` devuelve todas las filas de la bbdd
`raw=True` el cursor no realizará las conversiones automáticas entre tipos de datos
`dictionary=True` devuelve las filas como diccionarios
`named_tuple=True` devuelve las filas como named tuples
`cursor_class` un argumento que se puede usar para indicar que subclase queremos usar para instanciar el nuevo cursor

MySQL Connector/Python

Obtener resultados de una query

`variable_cursor.fetchone()` devuelve el primer resultado
`variable_cursor.fetchall()` devuelve todos los resultados como iterable – cada fila es una tupla

Pandas dataframe with SQL

`import pandas as pd`
`variable_df = pd.DataFrame(variable_resultado_fetchall, columns = ['columna1', 'columna2', ...])` crear un dataframe con los resultados de una query en una variable
`variable_df.head(n)` devuelve las n primeras filas del df, o 5 por defecto
`variable_df = pd.read_sql_query(variable_query, variable_cnx)` convertir los resultados de la query en df
`pd.read_sql(variable_query, variable_cnx)`
`variable_df.to_csv("nombre_archivo.csv")` guardar en csv
`variable_df.to_string()` formatear el dato en string
`variable_df.to_latex()` formatear el dato en un string que facilite la inserción en un documento latex

Crear y alterar una base de datos

`variable_cursor.execute("CREATE DATABASE nombre_BBDD")`
`variable_cursor.execute("CREATE TABLE nombre_tabla (nombre_columna TIPO, nombre_columna2 TIPO2)")`
`variable_cursor.execute("ALTER TABLE nombre_tabla ALTERACIONES")`

Insertar datos

`variable_query = "INSERT INTO nombre_tabla (columna1, columna2) VALUES (%s, %s)"`
`variable_valores = (valor1, valor2)`
`variable_cursor.execute(variable_query, variable_valores)`

otro método:

`variable_query = "UPDATE nombre_tabla SET nombre_columna = "nuevo_valor" WHERE nombre_columna = "valor"`

Insertar múltiples filas a una tabla

```
variable_valores_en_tuplas = ((valor1columna1, valor1columna2), (valor2columna1, valor2columna2))
variable_cursor.executemany(variable_query, variable_valores_en_tuplas)
```

`variable_conexion.commit()` después de ejecutar la inserción, para que los cambios efectúen en la BBDD
`variable_conexion.rollback()` se puede usar después de execute y antes de commit para deshacer los cambios
`print(variable_cursor.rowcount, "mensaje")` imprimir el número de filas en las cuales se han tomado la accion

Eliminar registros

`variable_query = "DROP TABLE nombre_tabla"`

Añadir errores

importar errorcode y usar try/except:

```
try:
    accion
except mysql.connector.Error as err:
    print(err)
    print("Error Code:", err.errno)
    print("SQLSTATE", err.sqlstate)
    print("Message", err.msg)
```