























APPENDIX

// HOME.HTML

{% load static %}

<!DOCTYPE html>

<**html lang="en"**>

<**head**>

<**meta charset="UTF-8"**>

<**title**>Home</**title**>

<**style type = text/css**>

div{

color:white;

}

h1{

color: 'white';

font-family: arial, sans-serif;

font-size: 60px;

font-weight: bold;

margin-top: 200px;

}

h2{

color: 'white';

font-family: arial, sans-serif;

font-size: 15px;

font-weight: bold;

margin-top: 400px;

}

body {

background-image: url("{% static 'DProject/images/img1.jpg' %}");

background-repeat: no-repeat;

background-attachment: fixed;

background-size: cover;

}

input[type=submit]{

background-color: #4dc3ff;

border: 2px;

color: white;

padding: 16px 32px;

cursor: pointer;

margin-top: 15px;

}

body{

background-image:url("{% static 'DiabetesPrediction/images/lifeline images.jpg' %}")

}

</**style**>

</**head**>

<**body**>

<**div align = "center"**>

<**h1**>

DIABETES PREDICTION SYSTEM

</**h1**>

<**form action = "predict"**>

<**input type = "submit" value = "Get Started"**>

</**form**>

</**div**>

</**body**>

</**html**>

//PREDICT.HTML

{%load static%}

<!DOCTYPE **html**>

<**html lang="en"**>

<**head**>

<**meta charset="UTF-8"**>

<**title**>Prediction Page</**title**>

<**style**>

body{

background-image : url(" {% static 'DiabetesPrediction/images/Predict page image.jpg' %}");

background-repeat:no-repeat;

background-attachment:fixed;

background-size:cover;

}

.main{

position:fixed;

top:140px;

left:410px;

width: 550px;

background-color:#ffffff;

border-radius: 10px;

align-items: center;

padding: 5%;

}

h1{

color: #0086b3;

font-size: 30px;

font-weight: bold;

}

input[type=submit]{

background-color: #4dc3ff;

border: 2px;

color: white;

padding: 8px 16px;

cursor: pointer;

margin-top: 15px;

}

input[type=submit]{

background-color: #4dc3ff;

border: 2px;

color: white;

padding: 8px 16px;

cursor: pointer;

margin-top: 15px;

}

</**style**>

</**head**>

<**body**>

<**p**>Prediction page!</**p**>

<**div align = 'center' class="main"**>

<**h1**>Please enter the following information:</**h1**>

<**form action="result"**>

<**table**>

<**tr**>

<**td align="right"**>Pregnancies:</**td**>

<**td align="left"**><**input type="text" name="n1"**></**td**>

</**tr**>

<**tr**>

<**td align="right"**>Glucose:</**td**>

<**td align="left"**><**input type="text" name="n2"**></**td**>

</**tr**>

<**tr**>

<**td align="right"**>Blood Pressure:</**td**>

<**td align="left"**><**input type="text" name="n3"**></**td**>

</**tr**>

<**tr**>

<**td align="right"**>Skin Thickness:</**td**>

<**td align="left"**><**input type="text" name="n4"**></**td**>

</**tr**>

<**tr**>

<**td align="right"**>Insulin:</**td**>

<**td align="left"**><**input type="text" name="n5"**></**td**>

</**tr**>

<**tr**>

<**td align="right"**>BMI:</**td**>

<**td align="left"**><**input type="text" name="n6"**></**td**>

</**tr**>

<**tr**>

<**td align="right"**>Diabetes Pedigree function:</**td**>

<**td align="left"**><**input type="text" name="n7"**></**td**>

</**tr**>

<**tr**>

<**td align="right"**>Age:</**td**>

<**td align="left"**><**input type="text" name="n8"**></**td**>

</**tr**>

</**table**>

<**input type="submit"**>

</**form**>

Result:{{result2}}

<**form action="knowmore"**>

<**input type="submit" value="Know More"**>

</**form**>

</**div**>

</**body**>

</**html**>

//KNOWMORE.HTML

{% load static %}

<!DOCTYPE **html**>

<**html lang="en"**>

<**head**>

<**meta charset="UTF-8"**>

<**title**>know more</**title**>

<**style**>

body{

background-image : url(" {% static 'DiabetesPrediction/images/medical image 6.jpg' %}");

background-repeat:no-repeat;

background-attachment:fixed;

background-size:cover;

font-size: 25px;

font-weight: bold;

align-items: center;

top:140px;

left:410px;

padding: 15%;

align-items: center;

}

.main{

position:fixed;

top:140px;

left:410px;

width: 550px;

background-color:#ffffff;

border-radius: 10px;

align-items: center;

padding: 5%;

border: 6px solid transparent;

}

h1{

color: #0086b3;

font-size: 40px;

font-weight: bold;

}

button {

background-color: #00BFFF;

border: none;

color: Black;

padding: 15px 32px;

text-align: center;

text-decoration: none;

display: inline-block;

font-size: 16px;

}

label {

color: #0086b3;

}

select

{

font-size: 20px;

}

input

{

font-size : 20px;

}

</**style**>

</**head**>

<**body**>

<**div align = 'center' class="main"**>

<**h1**>Select your age category</**h1**>

<**form action="knowmore"**>

<**label id = "Diabetes"**> Age :</**label**>

<**select onchange="showInfo(this.value)"**>

<**option value=""**>Select an age group</**option**>

<**option value="Young Adults"**>21 - 31</**option**>

<**option value="Adults"**>32 - 42</**option**>

<**option value="Middle - aged Adults"**>43 - 53</**option**>

<**option value="Old Adults"**>Above 53</**option**>

</**select**>

<**br**><**br**>

</**form**>

<**input id ="number" type="text"**>

<**button onclick="myFunction()"**>Submit</**button**>

<**p id = "tip"**></**p**>

<**script**>

function myFunction()

{

var res, Age;

Age = document.getElementById("number").value;

if ( Age >= 21 && Age <= 31)

{

res = "Avoid your risk by practising The Plate Method and healthy workout every day";

<!-- let aTag = document.createElement('a');-->

<!-- aTag.innerHTML="The Plate Method";-->

<!-- aTag.href="https://www.cdc.gov/diabetes/managing/eat-well/meal-plan-method.html";-->

<!-- document.body.appendChild(aTag);-->

<!-- aTag.style.textAlign = "center";-->

}

else if(Age >= 32 && Age <= 42)

{

res = "Avoid your risk by practising The Dash Chat Method and healthy workout every day";

<!-- align = "center ";-->

<!-- document.write('<a href="https://www.medicalnewstoday.com/articles/318277#considerations" title="Dash Chat Method">Dash Chat Method</a>')-->

}

else if (Age >= 43 && Age <= 53)

{

res = "By reducing 5% - 10 % of your total weight can help to lower your blood sugar level!";

}

else if (Age >= 53)

{

res = "A healthy walk and practicing 6 rules can lower your blood sugar level!";

}

else

{

res = "INVALID AGE";

}

document.getElementById("tip").innerHTML = res;

}

</**script**>

</**div**>

</**body**>

</**html**>

//VIEWS.PY

**from** django.shortcuts **import** render

**import** pandas **as** pd

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.ensemble **import** RandomForestClassifier

**def** home(request):

**return** render(request, **"home.html"**)

**def** predict(request):

**return** render(request, **"predict.html"**)

**def** knowmore(request):

**return** render(request, **"knowmore.html"**)

**def** result(request):

*# data = pd.read\_csv(r"C:\Users\divya\OneDrive\Desktop\diabetes.csv")*

data = pd.read\_csv(**r"C:\Users\divya\Downloads\diabetes\_new\_dataset - Sheet3.csv"**)

X = data.drop(**'Outcome'**, axis=1)

y = data[**'Outcome'**]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

rf = RandomForestClassifier()

rf.fit(X\_train, y\_train)

val1 = float(request.GET[**'n1'**])

val2 = float(request.GET[**'n2'**])

val3 = float(request.GET[**'n3'**])

val4 = float(request.GET[**'n4'**])

val5 = float(request.GET[**'n5'**])

val6 = float(request.GET[**'n6'**])

val7 = float(request.GET[**'n7'**])

val8 = float(request.GET[**'n8'**])

predictions = rf.predict([[val1, val2, val3, val4, val5, val6, val7, val8]])

*# predictions = rf.predict(data)*

result1 = **""**

**if** predictions == [1]:

result1 = **"Positive"**

**else**:

result1 = **"Negative"**

**return** render(request, **"predict.html"**, {**"result2"** : result1})

*# def knowmore(request):*

*#*

*# age =*

*#*

*# Tips1 = ""*

*#*

*# if age == "Children":*

*# Tips1 = "Avoid your risk by practising ""The Plate Method"" and healthy workout everyday"*

*# elif age == "Young Adults":*

*# Tips1 = "maon"*

*# elif age == "Adults":*

*# Tips1 = "absf"*

*# elif age == "Middle - aged Adults":*

*# Tips1 = "mnjn"*

*# elif age == "Old Adults":*

*# Tips1 = "bugej"*

*# else:*

*# Tips1 = "undefined"*

*# return render(request, "knowmore.html", {"tips2" : Tips1})*

//URLS.PY

**from** django.contrib **import** admin

**from** django.urls **import** path

**from** . **import** views

urlpatterns = [

path(**'admin/'**, admin.site.urls),

path(**""**, views.home),

path(**"predict/"**, views.predict),

path(**"predict/result"**, views.result),

path(**"predict/knowmore"**, views.knowmore),

]

//SETTINGS.PY

*"""*

*Django settings for prediction project.*

*Generated by 'django-admin startproject' using Django 3.2.5.*

*For more information on this file, see*

*https://docs.djangoproject.com/en/3.2/topics/settings/*

*For the full list of settings and their values, see*

*https://docs.djangoproject.com/en/3.2/ref/settings/*

*"""*

**import** os

**from** pathlib **import** Path

*# Build paths inside the project like this: BASE\_DIR / 'subdir'.*

BASE\_DIR = Path(\_\_file\_\_).resolve().parent.parent

*# Quick-start development settings - unsuitable for production*

*# See https://docs.djangoproject.com/en/3.2/howto/deployment/checklist/*

*# SECURITY WARNING: keep the secret key used in production secret!*

SECRET\_KEY = **'django-insecure-1&4b&q6e(7528iv\*av-w=xx)k-&2%zf(l7qj56mh&sp^ja86t-'**

*# SECURITY WARNING: don't run with debug turned on in production!*

DEBUG = **True**

ALLOWED\_HOSTS = []

*# Application definition*

INSTALLED\_APPS = [

**'django.contrib.admin'**,

**'django.contrib.auth'**,

**'django.contrib.contenttypes'**,

**'django.contrib.sessions'**,

**'django.contrib.messages'**,

**'django.contrib.staticfiles'**,

]

MIDDLEWARE = [

**'django.middleware.security.SecurityMiddleware'**,

**'django.contrib.sessions.middleware.SessionMiddleware'**,

**'django.middleware.common.CommonMiddleware'**,

**'django.middleware.csrf.CsrfViewMiddleware'**,

**'django.contrib.auth.middleware.AuthenticationMiddleware'**,

**'django.contrib.messages.middleware.MessageMiddleware'**,

**'django.middleware.clickjacking.XFrameOptionsMiddleware'**,

]

ROOT\_URLCONF = **'prediction.urls'**

TEMPLATES = [

{

**'BACKEND'**: **'django.template.backends.django.DjangoTemplates'**,

**'DIRS'**: [os.path.join(BASE\_DIR, **'templates'**)],

**'APP\_DIRS'**: **True**,

**'OPTIONS'**: {

**'context\_processors'**: [

**'django.template.context\_processors.debug'**,

**'django.template.context\_processors.request'**,

**'django.contrib.auth.context\_processors.auth'**,

**'django.contrib.messages.context\_processors.messages'**,

],

},

},

]

WSGI\_APPLICATION = **'prediction.wsgi.application'**

*# Database*

*# https://docs.djangoproject.com/en/3.2/ref/settings/#databases*

DATABASES = {

**'default'**: {

**'ENGINE'**: **'django.db.backends.sqlite3'**,

**'NAME'**: BASE\_DIR / **'db.sqlite3'**,

}

}

*# Password validation*

*# https://docs.djangoproject.com/en/3.2/ref/settings/#auth-password-validators*

AUTH\_PASSWORD\_VALIDATORS = [

{

**'NAME'**: **'django.contrib.auth.password\_validation.UserAttributeSimilarityValidator'**,

},

{

**'NAME'**: **'django.contrib.auth.password\_validation.MinimumLengthValidator'**,

},

{

**'NAME'**: **'django.contrib.auth.password\_validation.CommonPasswordValidator'**,

},

{

**'NAME'**: **'django.contrib.auth.password\_validation.NumericPasswordValidator'**,

},

]

*# Internationalization*

*# https://docs.djangoproject.com/en/3.2/topics/i18n/*

LANGUAGE\_CODE = **'en-us'**

TIME\_ZONE = **'UTC'**

USE\_I18N = **True**

USE\_L10N = **True**

USE\_TZ = **True**

*# Static files (CSS, JavaScript, Images)*

*# https://docs.djangoproject.com/en/3.2/howto/static-files/*

STATIC\_URL = **'/static/'**

STATICFILES\_DIRS = (

os.path.join(BASE\_DIR, **'static'**),

)

STATIC\_ROOT = os.path.join(os.path.dirname(BASE\_DIR), **"static"**)

*# Default primary key field type*

*# https://docs.djangoproject.com/en/3.2/ref/settings/#default-auto-field*

DEFAULT\_AUTO\_FIELD = **'django.db.models.BigAutoField'**

//MANAGE.PY

*"""Django's command-line utility for administrative tasks."""*

import os

import sys

def main():

*"""Run administrative tasks."""*

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'prediction.settings')

try:

from django.core.management import execute\_from\_command\_line

except ImportError as exc:

raise ImportError(

"Couldn't import Django. Are you sure it's installed and "

"available on your PYTHONPATH environment variable? Did you "

"forget to activate a virtual environment?"

) from exc

execute\_from\_command\_line(sys.argv)

if \_\_name\_\_ == '\_\_main\_\_':

main()

//WSGI.PY

*"""*

*WSGI config for prediction project.*

*It exposes the WSGI callable as a module-level variable named ``application``.*

*For more information on this file, see*

*https://docs.djangoproject.com/en/3.2/howto/deployment/wsgi/*

*"""*

**import** os

**from** django.core.wsgi **import** get\_wsgi\_application

os.environ.setdefault(**'DJANGO\_SETTINGS\_MODULE'**, **'prediction.settings'**)

application = get\_wsgi\_application()

//CODECS.PY

# encoding: utf-8

# module \_codecs

# from (built-in)

# by generator 1.147

# no doc

# no imports

# functions

def ascii\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def ascii\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def charmap\_build(\*args, \*\*kwargs): # real signature unknown

pass

def charmap\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def charmap\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def code\_page\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def code\_page\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def decode(\*args, \*\*kwargs): # real signature unknown

"""

Decodes obj using the codec registered for encoding.

Default encoding is 'utf-8'. errors may be given to set a

different error handling scheme. Default is 'strict' meaning that encoding

errors raise a ValueError. Other possible values are 'ignore', 'replace'

and 'backslashreplace' as well as any other name registered with

codecs.register\_error that can handle ValueErrors.

"""

pass

def encode(\*args, \*\*kwargs): # real signature unknown

"""

Encodes obj using the codec registered for encoding.

The default encoding is 'utf-8'. errors may be given to set a

different error handling scheme. Default is 'strict' meaning that encoding

errors raise a ValueError. Other possible values are 'ignore', 'replace'

and 'backslashreplace' as well as any other name registered with

codecs.register\_error that can handle ValueErrors.

"""

pass

def escape\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def escape\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def latin\_1\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def latin\_1\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def lookup(\*args, \*\*kwargs): # real signature unknown

""" Looks up a codec tuple in the Python codec registry and returns a CodecInfo object. """

pass

def lookup\_error(errors): # real signature unknown; restored from \_\_doc\_\_

"""

lookup\_error(errors) -> handler

Return the error handler for the specified error handling name or raise a

LookupError, if no handler exists under this name.

"""

pass

def mbcs\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def mbcs\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def oem\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def oem\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def raw\_unicode\_escape\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def raw\_unicode\_escape\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def readbuffer\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def register(\*args, \*\*kwargs): # real signature unknown

"""

Register a codec search function.

Search functions are expected to take one argument, the encoding name in

all lower case letters, and either return None, or a tuple of functions

(encoder, decoder, stream\_reader, stream\_writer) (or a CodecInfo object).

"""

pass

def register\_error(\*args, \*\*kwargs): # real signature unknown

"""

Register the specified error handler under the name errors.

handler must be a callable object, that will be called with an exception

instance containing information about the location of the encoding/decoding

error and must return a (replacement, new position) tuple.

"""

pass

def unicode\_escape\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def unicode\_escape\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_16\_be\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_16\_be\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_16\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_16\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_16\_ex\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_16\_le\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_16\_le\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_32\_be\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_32\_be\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_32\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_32\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_32\_ex\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_32\_le\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_32\_le\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_7\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_7\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_8\_decode(\*args, \*\*kwargs): # real signature unknown

pass

def utf\_8\_encode(\*args, \*\*kwargs): # real signature unknown

pass

def \_forget\_codec(\*args, \*\*kwargs): # real signature unknown

""" Purge the named codec from the internal codec lookup cache """

pass

# classes

class \_\_loader\_\_(object):

"""

Meta path import for built-in modules.

All methods are either class or static methods to avoid the need to

instantiate the class.

"""

@classmethod

def create\_module(cls, \*args, \*\*kwargs): # real signature unknown

""" Create a built-in module """

pass

@classmethod

def exec\_module(cls, \*args, \*\*kwargs): # real signature unknown

""" Exec a built-in module """

pass

@classmethod

def find\_module(cls, \*args, \*\*kwargs): # real signature unknown

"""

Find the built-in module.

If 'path' is ever specified then the search is considered a failure.

This method is deprecated. Use find\_spec() instead.

"""

pass

@classmethod

def find\_spec(cls, \*args, \*\*kwargs): # real signature unknown

pass

@classmethod

def get\_code(cls, \*args, \*\*kwargs): # real signature unknown

""" Return None as built-in modules do not have code objects. """

pass

@classmethod

def get\_source(cls, \*args, \*\*kwargs): # real signature unknown

""" Return None as built-in modules do not have source code. """

pass

@classmethod

def is\_package(cls, \*args, \*\*kwargs): # real signature unknown

""" Return False as built-in modules are never packages. """

pass

@classmethod

def load\_module(cls, \*args, \*\*kwargs): # real signature unknown

"""

Load the specified module into sys.modules and return it.

This method is deprecated. Use loader.exec\_module instead.

"""

pass

def module\_repr(module): # reliably restored by inspect

"""

Return repr for the module.

The method is deprecated. The import machinery does the job itself.

"""

pass

def \_\_init\_\_(self, \*args, \*\*kwargs): # real signature unknown

pass

\_\_weakref\_\_ = property(lambda self: object(), lambda self, v: None, lambda self: None) # default

"""list of weak references to the object (if defined)"""

\_ORIGIN = 'built-in'

\_\_dict\_\_ = None # (!) real value is "mappingproxy({'\_\_module\_\_': '\_frozen\_importlib', '\_\_doc\_\_': 'Meta path import for built-in modules.\\n\\n All methods are either class or static methods to avoid the need to\\n instantiate the class.\\n\\n ', '\_ORIGIN': 'built-in', 'module\_repr': <staticmethod object at 0x00000154E4926460>, 'find\_spec': <classmethod object at 0x00000154E4926490>, 'find\_module': <classmethod object at 0x00000154E49264C0>, 'create\_module': <classmethod object at 0x00000154E49264F0>, 'exec\_module': <classmethod object at 0x00000154E4926520>, 'get\_code': <classmethod object at 0x00000154E49265B0>, 'get\_source': <classmethod object at 0x00000154E4926640>, 'is\_package': <classmethod object at 0x00000154E49266D0>, 'load\_module': <classmethod object at 0x00000154E4926700>, '\_\_dict\_\_': <attribute '\_\_dict\_\_' of 'BuiltinImporter' objects>, '\_\_weakref\_\_': <attribute '\_\_weakref\_\_' of 'BuiltinImporter' objects>})"

# variables with complex values

\_\_spec\_\_ = None # (!) real value is "ModuleSpec(name='\_codecs', loader=<class '\_frozen\_importlib.BuiltinImporter'>, origin='built-in')"

//SQLITE3.PY

*# encoding: utf-8*

*# module \_sqlite3*

*# from C:\Users\divya\anaconda3\DLLs\\_sqlite3.pyd*

*# by generator 1.147*

*# no doc*

*# imports*

**import** sqlite3 **as** \_\_sqlite3

*# Variables with simple values*

PARSE\_COLNAMES = 2

PARSE\_DECLTYPES = 1

SQLITE\_ALTER\_TABLE = 26

SQLITE\_ANALYZE = 28

SQLITE\_ATTACH = 24

SQLITE\_CREATE\_INDEX = 1

SQLITE\_CREATE\_TABLE = 2

SQLITE\_CREATE\_TEMP\_INDEX = 3

SQLITE\_CREATE\_TEMP\_TABLE = 4

SQLITE\_CREATE\_TEMP\_TRIGGER = 5

SQLITE\_CREATE\_TEMP\_VIEW = 6

SQLITE\_CREATE\_TRIGGER = 7

SQLITE\_CREATE\_VIEW = 8

SQLITE\_CREATE\_VTABLE = 29

SQLITE\_DELETE = 9

SQLITE\_DENY = 1

SQLITE\_DETACH = 25

SQLITE\_DONE = 101

SQLITE\_DROP\_INDEX = 10

SQLITE\_DROP\_TABLE = 11

SQLITE\_DROP\_TEMP\_INDEX = 12

SQLITE\_DROP\_TEMP\_TABLE = 13

SQLITE\_DROP\_TEMP\_TRIGGER = 14

SQLITE\_DROP\_TEMP\_VIEW = 15

SQLITE\_DROP\_TRIGGER = 16

SQLITE\_DROP\_VIEW = 17

SQLITE\_DROP\_VTABLE = 30

SQLITE\_FUNCTION = 31

SQLITE\_IGNORE = 2

SQLITE\_INSERT = 18

SQLITE\_OK = 0

SQLITE\_PRAGMA = 19

SQLITE\_READ = 20

SQLITE\_RECURSIVE = 33

SQLITE\_REINDEX = 27

SQLITE\_SAVEPOINT = 32

SQLITE\_SELECT = 21

SQLITE\_TRANSACTION = 22

SQLITE\_UPDATE = 23

sqlite\_version = **'3.39.3'**

version = **'2.6.0'**

*# functions*

**def** adapt(obj, protocol, alternate): *# real signature unknown; restored from \_\_doc\_\_*

*""" adapt(obj, protocol, alternate) -> adapt obj to given protocol. """*

**pass**

**def** complete\_statement(sql): *# real signature unknown; restored from \_\_doc\_\_*

*"""*

*complete\_statement(sql)*

*Checks if a string contains a complete SQL statement.*

*"""*

**pass**

**def** connect(database, timeout=**None**, detect\_types=**None**, isolation\_level=**None**, check\_same\_thread=**None**, factory=**None**, cached\_statements=**None**, uri=**None**): *# real signature unknown; restored from \_\_doc\_\_*

*"""*

*connect(database[, timeout, detect\_types, isolation\_level,*

*check\_same\_thread, factory, cached\_statements, uri])*

*Opens a connection to the SQLite database file \*database\*. You can use*

*":memory:" to open a database connection to a database that resides in*

*RAM instead of on disk.*

*"""*

**pass**

**def** enable\_callback\_tracebacks(flag): *# real signature unknown; restored from \_\_doc\_\_*

*"""*

*enable\_callback\_tracebacks(flag)*

*Enable or disable callback functions throwing errors to stderr.*

*"""*

**pass**

**def** enable\_shared\_cache(do\_enable): *# real signature unknown; restored from \_\_doc\_\_*

*"""*

*enable\_shared\_cache(do\_enable)*

*Enable or disable shared cache mode for the calling thread.*

*"""*

**pass**

**def** register\_adapter(type, callable): *# real signature unknown; restored from \_\_doc\_\_*

*"""*

*register\_adapter(type, callable)*

*Registers an adapter with sqlite3's adapter registry.*

*"""*

**pass**

**def** register\_converter(typename, callable): *# real signature unknown; restored from \_\_doc\_\_*

*"""*

*register\_converter(typename, callable)*

*Registers a converter with sqlite3.*

*"""*

**pass**

*# classes*

**class** Connection(object):

*""" SQLite database connection object. """*

**def** backup(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Makes a backup of the database. """*

**pass**

**def** close(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Closes the connection. """*

**pass**

**def** commit(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Commit the current transaction. """*

**pass**

**def** create\_aggregate(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Creates a new aggregate. """*

**pass**

**def** create\_collation(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Creates a collation function. """*

**pass**

**def** create\_function(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Creates a new function. """*

**pass**

**def** cursor(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return a cursor for the connection. """*

**pass**

**def** enable\_load\_extension(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Enable dynamic loading of SQLite extension modules. """*

**pass**

**def** execute(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Executes an SQL statement. """*

**pass**

**def** executemany(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Repeatedly executes an SQL statement. """*

**pass**

**def** executescript(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Executes a multiple SQL statements at once. """*

**pass**

**def** interrupt(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Abort any pending database operation. """*

**pass**

**def** iterdump(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Returns iterator to the dump of the database in an SQL text format. """*

**pass**

**def** load\_extension(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Load SQLite extension module. """*

**pass**

**def** rollback(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Roll back the current transaction. """*

**pass**

**def** set\_authorizer(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Sets authorizer callback. """*

**pass**

**def** set\_progress\_handler(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Sets progress handler callback. """*

**pass**

**def** set\_trace\_callback(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Sets a trace callback called for each SQL statement (passed as unicode). """*

**pass**

**def** \_\_call\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Call self as a function. """*

**pass**

**def** \_\_enter\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" For context manager. """*

**pass**

**def** \_\_exit\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" For context manager. """*

**pass**

**def** \_\_init\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

**pass**

@staticmethod *# known case of \_\_new\_\_*

**def** \_\_new\_\_(\*args, \*\*kwargs): *# real signature unknown*

*""" Create and return a new object. See help(type) for accurate signature. """*

**pass**

DatabaseError = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

DataError = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

Error = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

IntegrityError = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

InterfaceError = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

InternalError = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

in\_transaction = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

isolation\_level = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

NotSupportedError = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

OperationalError = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

ProgrammingError = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

row\_factory = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

text\_factory = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

total\_changes = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

Warning = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

**class** Cursor(object):

*""" SQLite database cursor class. """*

**def** close(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Closes the cursor. """*

**pass**

**def** execute(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Executes an SQL statement. """*

**pass**

**def** executemany(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Repeatedly executes an SQL statement. """*

**pass**

**def** executescript(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Executes multiple SQL statements at once. """*

**pass**

**def** fetchall(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Fetches all rows from the resultset. """*

**pass**

**def** fetchmany(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Fetches several rows from the resultset. """*

**pass**

**def** fetchone(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Fetches one row from the resultset. """*

**pass**

**def** setinputsizes(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Required by DB-API. Does nothing in sqlite3. """*

**pass**

**def** setoutputsize(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Required by DB-API. Does nothing in sqlite3. """*

**pass**

**def** \_\_init\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

**pass**

**def** \_\_iter\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Implement iter(self). """*

**pass**

@staticmethod *# known case of \_\_new\_\_*

**def** \_\_new\_\_(\*args, \*\*kwargs): *# real signature unknown*

*""" Create and return a new object. See help(type) for accurate signature. """*

**pass**

**def** \_\_next\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Implement next(self). """*

**pass**

arraysize = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

connection = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

description = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

lastrowid = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

rowcount = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

row\_factory = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

**class** Error(Exception):

*# no doc*

**def** \_\_init\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

**pass**

\_\_weakref\_\_ = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

**"""list of weak references to the object (if defined)"""**

**class** DatabaseError(\_\_sqlite3.Error):

*# no doc*

**def** \_\_init\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

**pass**

**class** DataError(\_\_sqlite3.DatabaseError):

*# no doc*

**def** \_\_init\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

**pass**

**class** IntegrityError(\_\_sqlite3.DatabaseError):

*# no doc*

**def** \_\_init\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

**pass**

**class** InterfaceError(\_\_sqlite3.Error):

*# no doc*

**def** \_\_init\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

**pass**

**class** InternalError(\_\_sqlite3.DatabaseError):

*# no doc*

**def** \_\_init\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

**pass**

**class** NotSupportedError(\_\_sqlite3.DatabaseError):

*# no doc*

**def** \_\_init\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

**pass**

**class** OperationalError(\_\_sqlite3.DatabaseError):

*# no doc*

**def** \_\_init\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

**pass**

**class** OptimizedUnicode(object):

*"""*

*str(object='') -> str*

*str(bytes\_or\_buffer[, encoding[, errors]]) -> str*

*Create a new string object from the given object. If encoding or*

*errors is specified, then the object must expose a data buffer*

*that will be decoded using the given encoding and error handler.*

*Otherwise, returns the result of object.\_\_str\_\_() (if defined)*

*or repr(object).*

*encoding defaults to sys.getdefaultencoding().*

*errors defaults to 'strict'.*

*"""*

**def** capitalize(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return a capitalized version of the string.*

*More specifically, make the first character have upper case and the rest lower*

*case.*

*"""*

**pass**

**def** casefold(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return a version of the string suitable for caseless comparisons. """*

**pass**

**def** center(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return a centered string of length width.*

*Padding is done using the specified fill character (default is a space).*

*"""*

**pass**

**def** count(self, sub, start=**None**, end=**None**): *# real signature unknown; restored from \_\_doc\_\_*

*"""*

*S.count(sub[, start[, end]]) -> int*

*Return the number of non-overlapping occurrences of substring sub in*

*string S[start:end]. Optional arguments start and end are*

*interpreted as in slice notation.*

*"""*

**return** 0

**def** encode(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Encode the string using the codec registered for encoding.*

*encoding*

*The encoding in which to encode the string.*

*errors*

*The error handling scheme to use for encoding errors.*

*The default is 'strict' meaning that encoding errors raise a*

*UnicodeEncodeError. Other possible values are 'ignore', 'replace' and*

*'xmlcharrefreplace' as well as any other name registered with*

*codecs.register\_error that can handle UnicodeEncodeErrors.*

*"""*

**pass**

**def** endswith(self, suffix, start=**None**, end=**None**): *# real signature unknown; restored from \_\_doc\_\_*

*"""*

*S.endswith(suffix[, start[, end]]) -> bool*

*Return True if S ends with the specified suffix, False otherwise.*

*With optional start, test S beginning at that position.*

*With optional end, stop comparing S at that position.*

*suffix can also be a tuple of strings to try.*

*"""*

**return False**

**def** expandtabs(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return a copy where all tab characters are expanded using spaces.*

*If tabsize is not given, a tab size of 8 characters is assumed.*

*"""*

**pass**

**def** find(self, sub, start=**None**, end=**None**): *# real signature unknown; restored from \_\_doc\_\_*

*"""*

*S.find(sub[, start[, end]]) -> int*

*Return the lowest index in S where substring sub is found,*

*such that sub is contained within S[start:end]. Optional*

*arguments start and end are interpreted as in slice notation.*

*Return -1 on failure.*

*"""*

**return** 0

**def** format(self, \*args, \*\*kwargs): *# real signature unknown; restored from \_\_doc\_\_*

*"""*

*S.format(\*args, \*\*kwargs) -> str*

*Return a formatted version of S, using substitutions from args and kwargs.*

*The substitutions are identified by braces ('{' and '}').*

*"""*

**return ""**

**def** format\_map(self, mapping): *# real signature unknown; restored from \_\_doc\_\_*

*"""*

*S.format\_map(mapping) -> str*

*Return a formatted version of S, using substitutions from mapping.*

*The substitutions are identified by braces ('{' and '}').*

*"""*

**return ""**

**def** index(self, sub, start=**None**, end=**None**): *# real signature unknown; restored from \_\_doc\_\_*

*"""*

*S.index(sub[, start[, end]]) -> int*

*Return the lowest index in S where substring sub is found,*

*such that sub is contained within S[start:end]. Optional*

*arguments start and end are interpreted as in slice notation.*

*Raises ValueError when the substring is not found.*

*"""*

**return** 0

**def** isalnum(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return True if the string is an alpha-numeric string, False otherwise.*

*A string is alpha-numeric if all characters in the string are alpha-numeric and*

*there is at least one character in the string.*

*"""*

**pass**

**def** isalpha(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return True if the string is an alphabetic string, False otherwise.*

*A string is alphabetic if all characters in the string are alphabetic and there*

*is at least one character in the string.*

*"""*

**pass**

**def** isascii(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return True if all characters in the string are ASCII, False otherwise.*

*ASCII characters have code points in the range U+0000-U+007F.*

*Empty string is ASCII too.*

*"""*

**pass**

**def** isdecimal(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return True if the string is a decimal string, False otherwise.*

*A string is a decimal string if all characters in the string are decimal and*

*there is at least one character in the string.*

*"""*

**pass**

**def** isdigit(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return True if the string is a digit string, False otherwise.*

*A string is a digit string if all characters in the string are digits and there*

*is at least one character in the string.*

*"""*

**pass**

**def** isidentifier(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return True if the string is a valid Python identifier, False otherwise.*

*Call keyword.iskeyword(s) to test whether string s is a reserved identifier,*

*such as "def" or "class".*

*"""*

**pass**

**def** islower(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return True if the string is a lowercase string, False otherwise.*

*A string is lowercase if all cased characters in the string are lowercase and*

*there is at least one cased character in the string.*

*"""*

**pass**

**def** isnumeric(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return True if the string is a numeric string, False otherwise.*

*A string is numeric if all characters in the string are numeric and there is at*

*least one character in the string.*

*"""*

**pass**

**def** isprintable(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return True if the string is printable, False otherwise.*

*A string is printable if all of its characters are considered printable in*

*repr() or if it is empty.*

*"""*

**pass**

**def** isspace(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return True if the string is a whitespace string, False otherwise.*

*A string is whitespace if all characters in the string are whitespace and there*

*is at least one character in the string.*

*"""*

**pass**

**def** istitle(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return True if the string is a title-cased string, False otherwise.*

*In a title-cased string, upper- and title-case characters may only*

*follow uncased characters and lowercase characters only cased ones.*

*"""*

**pass**

**def** isupper(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return True if the string is an uppercase string, False otherwise.*

*A string is uppercase if all cased characters in the string are uppercase and*

*there is at least one cased character in the string.*

*"""*

**pass**

**def** join(self, ab=**None**, pq=**None**, rs=**None**): *# real signature unknown; restored from \_\_doc\_\_*

*"""*

*Concatenate any number of strings.*

*The string whose method is called is inserted in between each given string.*

*The result is returned as a new string.*

*Example: '.'.join(['ab', 'pq', 'rs']) -> 'ab.pq.rs'*

*"""*

**pass**

**def** ljust(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return a left-justified string of length width.*

*Padding is done using the specified fill character (default is a space).*

*"""*

**pass**

**def** lower(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return a copy of the string converted to lowercase. """*

**pass**

**def** lstrip(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return a copy of the string with leading whitespace removed.*

*If chars is given and not None, remove characters in chars instead.*

*"""*

**pass**

**def** maketrans(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return a translation table usable for str.translate().*

*If there is only one argument, it must be a dictionary mapping Unicode*

*ordinals (integers) or characters to Unicode ordinals, strings or None.*

*Character keys will be then converted to ordinals.*

*If there are two arguments, they must be strings of equal length, and*

*in the resulting dictionary, each character in x will be mapped to the*

*character at the same position in y. If there is a third argument, it*

*must be a string, whose characters will be mapped to None in the result.*

*"""*

**pass**

**def** partition(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Partition the string into three parts using the given separator.*

*This will search for the separator in the string. If the separator is found,*

*returns a 3-tuple containing the part before the separator, the separator*

*itself, and the part after it.*

*If the separator is not found, returns a 3-tuple containing the original string*

*and two empty strings.*

*"""*

**pass**

**def** removeprefix(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return a str with the given prefix string removed if present.*

*If the string starts with the prefix string, return string[len(prefix):].*

*Otherwise, return a copy of the original string.*

*"""*

**pass**

**def** removesuffix(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return a str with the given suffix string removed if present.*

*If the string ends with the suffix string and that suffix is not empty,*

*return string[:-len(suffix)]. Otherwise, return a copy of the original*

*string.*

*"""*

**pass**

**def** replace(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return a copy with all occurrences of substring old replaced by new.*

*count*

*Maximum number of occurrences to replace.*

*-1 (the default value) means replace all occurrences.*

*If the optional argument count is given, only the first count occurrences are*

*replaced.*

*"""*

**pass**

**def** rfind(self, sub, start=**None**, end=**None**): *# real signature unknown; restored from \_\_doc\_\_*

*"""*

*S.rfind(sub[, start[, end]]) -> int*

*Return the highest index in S where substring sub is found,*

*such that sub is contained within S[start:end]. Optional*

*arguments start and end are interpreted as in slice notation.*

*Return -1 on failure.*

*"""*

**return** 0

**def** rindex(self, sub, start=**None**, end=**None**): *# real signature unknown; restored from \_\_doc\_\_*

*"""*

*S.rindex(sub[, start[, end]]) -> int*

*Return the highest index in S where substring sub is found,*

*such that sub is contained within S[start:end]. Optional*

*arguments start and end are interpreted as in slice notation.*

*Raises ValueError when the substring is not found.*

*"""*

**return** 0

**def** rjust(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return a right-justified string of length width.*

*Padding is done using the specified fill character (default is a space).*

*"""*

**pass**

**def** rpartition(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Partition the string into three parts using the given separator.*

*This will search for the separator in the string, starting at the end. If*

*the separator is found, returns a 3-tuple containing the part before the*

*separator, the separator itself, and the part after it.*

*If the separator is not found, returns a 3-tuple containing two empty strings*

*and the original string.*

*"""*

**pass**

**def** rsplit(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return a list of the words in the string, using sep as the delimiter string.*

*sep*

*The delimiter according which to split the string.*

*None (the default value) means split according to any whitespace,*

*and discard empty strings from the result.*

*maxsplit*

*Maximum number of splits to do.*

*-1 (the default value) means no limit.*

*Splits are done starting at the end of the string and working to the front.*

*"""*

**pass**

**def** rstrip(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return a copy of the string with trailing whitespace removed.*

*If chars is given and not None, remove characters in chars instead.*

*"""*

**pass**

**def** split(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return a list of the words in the string, using sep as the delimiter string.*

*sep*

*The delimiter according which to split the string.*

*None (the default value) means split according to any whitespace,*

*and discard empty strings from the result.*

*maxsplit*

*Maximum number of splits to do.*

*-1 (the default value) means no limit.*

*"""*

**pass**

**def** splitlines(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return a list of the lines in the string, breaking at line boundaries.*

*Line breaks are not included in the resulting list unless keepends is given and*

*true.*

*"""*

**pass**

**def** startswith(self, prefix, start=**None**, end=**None**): *# real signature unknown; restored from \_\_doc\_\_*

*"""*

*S.startswith(prefix[, start[, end]]) -> bool*

*Return True if S starts with the specified prefix, False otherwise.*

*With optional start, test S beginning at that position.*

*With optional end, stop comparing S at that position.*

*prefix can also be a tuple of strings to try.*

*"""*

**return False**

**def** strip(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return a copy of the string with leading and trailing whitespace removed.*

*If chars is given and not None, remove characters in chars instead.*

*"""*

**pass**

**def** swapcase(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Convert uppercase characters to lowercase and lowercase characters to uppercase. """*

**pass**

**def** title(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Return a version of the string where each word is titlecased.*

*More specifically, words start with uppercased characters and all remaining*

*cased characters have lower case.*

*"""*

**pass**

**def** translate(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Replace each character in the string using the given translation table.*

*table*

*Translation table, which must be a mapping of Unicode ordinals to*

*Unicode ordinals, strings, or None.*

*The table must implement lookup/indexing via \_\_getitem\_\_, for instance a*

*dictionary or list. If this operation raises LookupError, the character is*

*left untouched. Characters mapped to None are deleted.*

*"""*

**pass**

**def** upper(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return a copy of the string converted to uppercase. """*

**pass**

**def** zfill(self, \*args, \*\*kwargs): *# real signature unknown*

*"""*

*Pad a numeric string with zeros on the left, to fill a field of the given width.*

*The string is never truncated.*

*"""*

**pass**

**def** \_\_add\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self+value. """*

**pass**

**def** \_\_contains\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return key in self. """*

**pass**

**def** \_\_eq\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self==value. """*

**pass**

**def** \_\_format\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return a formatted version of the string as described by format\_spec. """*

**pass**

**def** \_\_getattribute\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return getattr(self, name). """*

**pass**

**def** \_\_getitem\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self[key]. """*

**pass**

**def** \_\_getnewargs\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

**pass**

**def** \_\_ge\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self>=value. """*

**pass**

**def** \_\_gt\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self>value. """*

**pass**

**def** \_\_hash\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return hash(self). """*

**pass**

**def** \_\_init\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

**pass**

**def** \_\_iter\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Implement iter(self). """*

**pass**

**def** \_\_len\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return len(self). """*

**pass**

**def** \_\_le\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self<=value. """*

**pass**

**def** \_\_lt\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self<value. """*

**pass**

**def** \_\_mod\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self%value. """*

**pass**

**def** \_\_mul\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self\*value. """*

**pass**

@staticmethod *# known case of \_\_new\_\_*

**def** \_\_new\_\_(\*args, \*\*kwargs): *# real signature unknown*

*""" Create and return a new object. See help(type) for accurate signature. """*

**pass**

**def** \_\_ne\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self!=value. """*

**pass**

**def** \_\_repr\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return repr(self). """*

**pass**

**def** \_\_rmod\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return value%self. """*

**pass**

**def** \_\_rmul\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return value\*self. """*

**pass**

**def** \_\_sizeof\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return the size of the string in memory, in bytes. """*

**pass**

**def** \_\_str\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return str(self). """*

**pass**

**class** PrepareProtocol(object):

*# no doc*

**def** \_\_init\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

**pass**

@staticmethod *# known case of \_\_new\_\_*

**def** \_\_new\_\_(\*args, \*\*kwargs): *# real signature unknown*

*""" Create and return a new object. See help(type) for accurate signature. """*

**pass**

**class** ProgrammingError(\_\_sqlite3.DatabaseError):

*# no doc*

**def** \_\_init\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

**pass**

**class** Row(object):

*# no doc*

**def** keys(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Returns the keys of the row. """*

**pass**

**def** \_\_eq\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self==value. """*

**pass**

**def** \_\_getitem\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self[key]. """*

**pass**

**def** \_\_ge\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self>=value. """*

**pass**

**def** \_\_gt\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self>value. """*

**pass**

**def** \_\_hash\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return hash(self). """*

**pass**

**def** \_\_init\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

**pass**

**def** \_\_iter\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Implement iter(self). """*

**pass**

**def** \_\_len\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return len(self). """*

**pass**

**def** \_\_le\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self<=value. """*

**pass**

**def** \_\_lt\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self<value. """*

**pass**

@staticmethod *# known case of \_\_new\_\_*

**def** \_\_new\_\_(\*args, \*\*kwargs): *# real signature unknown*

*""" Create and return a new object. See help(type) for accurate signature. """*

**pass**

**def** \_\_ne\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

*""" Return self!=value. """*

**pass**

**class** Warning(Exception):

*# no doc*

**def** \_\_init\_\_(self, \*args, \*\*kwargs): *# real signature unknown*

**pass**

\_\_weakref\_\_ = property(**lambda** self: object(), **lambda** self, v: **None**, **lambda** self: **None**) *# default*

**"""list of weak references to the object (if defined)"""**

*# variables with complex values*

adapters = {

(

**None**, *# (!) real value is "<class 'datetime.date'>"*

PrepareProtocol,

):

**None** *# (!) real value is '<function register\_adapters\_and\_converters.<locals>.adapt\_date at 0x000001CD2FEF65E0>'*

,

(

**None**, *# (!) real value is "<class 'datetime.datetime'>"*

**'<value is a self-reference, replaced by this string>'**,

):

**None** *# (!) real value is '<function register\_adapters\_and\_converters.<locals>.adapt\_datetime at 0x000001CD2FEF6670>'*

,

}

converters = {

**'DATE'**: **None**, *# (!) real value is '<function register\_adapters\_and\_converters.<locals>.convert\_date at 0x000001CD2FEF6550>'*

**'TIMESTAMP'**: **None**, *# (!) real value is '<function register\_adapters\_and\_converters.<locals>.convert\_timestamp at 0x000001CD2FEF6280>'*

}

\_\_loader\_\_ = **None** *# (!) real value is '<\_frozen\_importlib\_external.ExtensionFileLoader object at 0x000001CD2F4A38B0>'*

\_\_spec\_\_ = **None** *# (!) real value is "ModuleSpec(name='\_sqlite3', loader=<\_frozen\_importlib\_external.ExtensionFileLoader object at 0x000001CD2F4A38B0>, origin='C:\\\\Users\\\\divya\\\\anaconda3\\\\DLLs\\\\\_sqlite3.pyd')"*