Introducción a la POO y APIs

Introducción a la Programación

¿Qué es la Programación Orientada a Objetos (POO)?

¿Qué es la Programación Orientada a Objetos (POO)?

- Es un paradigma de programación (como Funcional e Imperativo)

¿Qué es la Programación Orientada a Objetos (POO)?

- Es un paradigma de programación (como Funcional e Imperativo)
- Es pensar y organizar el código en términos de clases y objetos (también se los llama instancias de una clase)

¿Qué es la Programación Orientada a Objetos (POO)?

- Es un paradigma de programación (como Funcional e Imperativo)
- Es pensar y organizar el código en términos de clases y objetos (también se los llama instancias de una clase)
- Las clases son entidades que combinan variables (atributos) y operaciones (métodos) para definir las propiedades y el comportamiento de los objetos.

¿Qué es la Programación Orientada a Objetos (POO)?

- Es un paradigma de programación (como Funcional e Imperativo)
- Es pensar y organizar el código en términos de clases y objetos (también se los llama instancias de una clase)
- Las clases son entidades que combinan variables (atributos) y operaciones (métodos) para definir las propiedades y el comportamiento de los objetos.
- Las clases definen nuevos tipos de datos.

¿Qué es la Programación Orientada a Objetos (POO)?

- Es un paradigma de programación (como Funcional e Imperativo)
- Es pensar y organizar el código en términos de clases y objetos (también se los llama instancias de una clase)
- Las clases son entidades que combinan variables (atributos) y operaciones (métodos) para definir las propiedades y el comportamiento de los objetos.
- Las clases definen nuevos tipos de datos.
- Los objetos pueden interactuar entre sí a través de sus métodos y atributos.

¿Qué es la Programación Orientada a Objetos (POO)?

- Es un paradigma de programación (como Funcional e Imperativo)
- Es pensar y organizar el código en términos de clases y objetos (también se los llama instancias de una clase)
- Las clases son entidades que combinan variables (atributos) y operaciones (métodos) para definir las propiedades y el comportamiento de los objetos.
- Las clases definen nuevos tipos de datos.
- Los objetos pueden interactuar entre sí a través de sus métodos y atributos.
- Se asemeja a la forma en que pensamos y modelamos el mundo real mediante TADs, pero a nivel de lenguaje de programación, no de especificación.

¿Qué es la Programación Orientada a Objetos (POO)?

- Es un paradigma de programación (como Funcional e Imperativo)
- Es pensar y organizar el código en términos de clases y objetos (también se los llama instancias de una clase)
- Las clases son entidades que combinan variables (atributos) y operaciones (métodos) para definir las propiedades y el comportamiento de los objetos.
- Las clases definen nuevos tipos de datos.
- Los objetos pueden interactuar entre sí a través de sus métodos y atributos.
- Se asemeja a la forma en que pensamos y modelamos el mundo real mediante TADs, pero a nivel de lenguaje de programación, no de especificación.
- Algunos conceptos clave de la POO incluyen encapsulamiento, herencia y polimorfismo.

¿Qué es la Programación Orientada a Objetos (POO)?

- Es un paradigma de programación (como Funcional e Imperativo)
- Es pensar y organizar el código en términos de clases y objetos (también se los llama instancias de una clase)
- Las clases son entidades que combinan variables (atributos) y operaciones (métodos) para definir las propiedades y el comportamiento de los objetos.
- Las clases definen nuevos tipos de datos.
- Los objetos pueden interactuar entre sí a través de sus métodos y atributos.
- Se asemeja a la forma en que pensamos y modelamos el mundo real mediante TADs, pero a nivel de lenguaje de programación, no de especificación.
- Algunos conceptos clave de la POO incluyen encapsulamiento, herencia y polimorfismo.
- Python está fuertemente orientado a objetos, no obstante, no es condición necesaria hacer uso de las clases para crear un programa (como ocurre en otros lenguajes como Java o Smalltalk).

¿Qué es una Clase en Python?

- Definición de una clase en Pyhton

```
class Persona:
    def __init__(self, nombre: str, edad:int):
        self.nombre = nombre
        self.edad = edad

    def presentarse(self):
        print(f"Hola, mi nombre es {self.nombre} y mi edad
     es {self.edad}")
```

- `__init__` es un método especial o constructor que se utiliza para inicializar los objetos (instancias) de una clase.
- Este método se llama automáticamente cuando se crea un nuevo objeto.
- El primer parámetro de `__init__` es `self`, que es una referencia al propio objeto.
- Todos los métodos (funciones) definidos dentro de una clase deben tener el parámetro `self` a través del cual se pueden acceder y modificar los atributos del objeto.

¿Qué es una Clase en Python?

- Creación de una instancia de una clase (objeto)

```
# Crear una instancia de la clase Persona
persona1 = Persona("Juan", 30)
```

- Acceso a los atributos de un objeto

```
print(persona1.nombre)  # Imprime "Juan"
print(persona1.edad)    # Imprime 30
persona1.edad = 20      # Asigna 20 al atributo edad
```

- Uso de los metodos de la instancia

```
persona1.presentarse()
# Imprime "Hola, mi nombre es Juan y mi edad es 20"
```

Encapsulamiento

Encapsulamiento

- Permite ocultar los detalles internos de implementación de una clase.

Encapsulamiento

- Permite ocultar los detalles internos de implementación de una clase.
- Los usuarios de una clase solo necesitan conocer la interfaz pública, es decir, los métodos y atributos accesibles, respetando la documentación (especificación) de la clase.

Encapsulamiento

- Permite ocultar los detalles internos de implementación de una clase.
- Los usuarios de una clase solo necesitan conocer la interfaz pública, es decir, los métodos y atributos accesibles, respetando la documentación (especificación) de la clase.
- En Python, tanto los atributos como los métodos de una clase pueden tener tres niveles de visibilidad en términos de acceso desde fuera de la clase: público, protegido y privado.

Encapsulamiento

- Permite ocultar los detalles internos de implementación de una clase.
- Los usuarios de una clase solo necesitan conocer la interfaz pública, es decir, los métodos y atributos accesibles, respetando la documentación (especificación) de la clase.
- En Python, tanto los atributos como los métodos de una clase pueden tener tres niveles de visibilidad en términos de acceso desde fuera de la clase: público, protegido y privado.
- Estos niveles de visibilidad se indican mediante el uso de guiones bajos (*underscores*) en los nombres de los atributos y métodos.

Encapsulamiento

- Permite ocultar los detalles internos de implementación de una clase.
- Los usuarios de una clase solo necesitan conocer la interfaz pública, es decir, los métodos y atributos accesibles, respetando la documentación (especificación) de la clase.
- En Python, tanto los atributos como los métodos de una clase pueden tener tres niveles de visibilidad en términos de acceso desde fuera de la clase: público, protegido y privado.
- Estos niveles de visibilidad se indican mediante el uso de guiones bajos (*underscores*) en los nombres de los atributos y métodos.
- Para atributos o métodos protegidos se utiliza un guion bajo y para privados doble guión bajo como prefijo en sus nombres.

Encapsulamiento

```python
class Persona:
    def __init__(self, nombre: str, edad: int):
        self.__nombre = nombre
        self.__edad = edad

    def dameNombre(self):
        return self.__nombre

    def dameEdad(self):
        return self.__edad

    def definirNombre(self, nombre: str):
        self.__nombre = nombre

    def definirEdad(self, edad: int):
        self.__edad = edad

    def presentarse(self):
        print(f"Hola, mi nombre es {self.__nombre} y mi edad es {self.__edad}")

persona1 = Persona("Pablo", 30)
persona1.presentarse()
#print(persona1.__nombre) #Esto da error, no se puede acceder directamente
print(persona1.dameNombre())
persona1.definirNombre("Pepe")
print(persona1.dameNombre())
```

Herencia

Herencia

- Permite la creación de nuevas clases basadas en clases existentes, heredando sus atributos y métodos.

Herencia

- Permite la creación de nuevas clases basadas en clases existentes, heredando sus atributos y métodos.
- Las clases nuevas se llaman derivadas o subclases y la clase existente de la cual heredan se llama clase base o superclase.

Herencia

- Permite la creación de nuevas clases basadas en clases existentes, heredando sus atributos y métodos.
- Las clases nuevas se llaman derivadas o subclases y la clase existente de la cual heredan se llama clase base o superclase.
- Define jerarquías de clases que comparten diversos métodos y atributos.

Herencia

- Permite la creación de nuevas clases basadas en clases existentes, heredando sus atributos y métodos.
- Las clases nuevas se llaman derivadas o subclases y la clase existente de la cual heredan se llama clase base o superclase.
- Define jerarquías de clases que comparten diversos métodos y atributos.
- Podemos sobrecargar (sobre-escribir) métodos para cada subclase.

Herencia

```python
class Animal:

    def __init__(self, nombre: str , edad: int):
        self.nombre = nombre
        self.edad = edad

    def emitir_sonido(self):
        pass

class Perro(Animal):
    def emitir_sonido(self):
        return "Guau!"

class Gato(Animal):
    def emitir_sonido(self):
        return "Miau!"

class Pato(Animal):
    def emitir_sonido(self):
        return "Cuac!"

# Crear instancias de diferentes animales
miPerro = Perro("Dylan", 10)
miGato = Gato("Azrael", 15)
miPato = Pato("Daffy", 2)

# Acceder a los atributos y metodos de las subclases
print(f"{miPerro.nombre} tiene {miPerro.edad} y hace {miPerro.emitir_sonido()}")
print(f"{miGato.nombre} tiene {miGato.edad} y hace {miGato.emitir_sonido()}")
```

Polimorfismo

Polimorfismo

- Recordatorio: se llama polimorfismo a una función que puede aplicarse a distintos tipos de datos sin redefinirla.

Polimorfismo

- Recordatorio: se llama polimorfismo a una función que puede aplicarse a distintos tipos de datos sin redefinirla.
- En Python, el polimorfismo se logra a través de la herencia y la implementación de métodos con el mismo nombre en diferentes subclases.

Polimorfismo

- Recordatorio: se llama polimorfismo a una función que puede aplicarse a distintos tipos de datos sin redefinirla.
- En Python, el polimorfismo se logra a través de la herencia y la implementación de métodos con el mismo nombre en diferentes subclases.
- Mediante la sobrecarga de métodos puedo obtener una función que puede recibir distintos objetos (tipos de datos) sin redefinirla.

Polimorfismo

```python
class Animal:

    def __init__(self, nombre: str , edad: int):
        self.nombre = nombre
        self.edad = edad

    def emitir_sonido(self):
        pass

class Perro(Animal):
    def emitir_sonido(self):
        return "Guau!"

class Gato(Animal):
    def emitir_sonido(self):
        return "Miau!"

# Funcion que utiliza polimorfismo
def hacer_emitir_sonido(animal):
    return animal.emitir_sonido()

# Crear instancias de diferentes animales
miPerro = Perro("Dylan", 10)
miGato = Gato("Azrael", 15)

# Llamar a la funcion polimorfica con diferentes instancias
print(hacer_emitir_sonido(miPerro))  # Imprime "Guau!"
print(hacer_emitir_sonido(miGato))   # Imprime "Miau!"
```

Un paso más allá: ¿Qué es una API?

Un poquito fuera del alcance de la materia...

Un paso más allá: ¿Qué es una API?

Un poquito fuera del alcance de la materia...

- El término API es muy usado actualmente y está relacionado con poder usar desde un programa funcionalidades de otro programa.

Un paso más allá: ¿Qué es una API?

Un poquito fuera del alcance de la materia...

- El término API es muy usado actualmente y está relacionado con poder usar desde un programa funcionalidades de otro programa.
- API significa *Application Programming Interface* (Interfaz de Programación de Aplicaciones, en español). Una API define cómo las distintas partes de un software deben interactuar, especificando los métodos y formatos de datos que se utilizan para el intercambio de información.

Un paso más allá: ¿Qué es una API?

Un poquito fuera del alcance de la materia...

- El término API es muy usado actualmente y está relacionado con poder usar desde un programa funcionalidades de otro programa.
- API significa *Application Programming Interface* (Interfaz de Programación de Aplicaciones, en español). Una API define cómo las distintas partes de un software deben interactuar, especificando los métodos y formatos de datos que se utilizan para el intercambio de información.
- En el contexto de desarrollo de software, una API puede ser considerada como un contrato entre dos aplicaciones.

Un paso más allá: ¿Qué es una API?

Un poquito fuera del alcance de la materia...

- El término API es muy usado actualmente y está relacionado con poder usar desde un programa funcionalidades de otro programa.
- API significa *Application Programming Interface* (Interfaz de Programación de Aplicaciones, en español). Una API define cómo las distintas partes de un software deben interactuar, especificando los métodos y formatos de datos que se utilizan para el intercambio de información.
- En el contexto de desarrollo de software, una API puede ser considerada como un contrato entre dos aplicaciones.
- Una API encapsula el comportamiento de otro programa y en muchos casos, su utilización es similar al uso de un TAD. Detrás de este encapsulamiento se esconden un gran números de problemas a resolver como ser: conexiones de red, uso de protocolos, manejo de errores, transformaciones de datos, etc (y son muchos etcs).

Veamos una API cualquiera: Google Translate API

Un paso más allá: ¿Qué es una API?

- Instalamos el módulo: pip install googletrans googletrans==3.1.0a0
- Y veamos que nos ofrece su contrato:

translator = Translator()

- El método translate(texto, idioma origen, idioma destino) devuelve la siguiente estructura :
  - src: idioma original dest: idioma destino
  - origin: texto en idioma original
  - text: texto traducido
  - pronunciation: pronunciación del texto tranducido

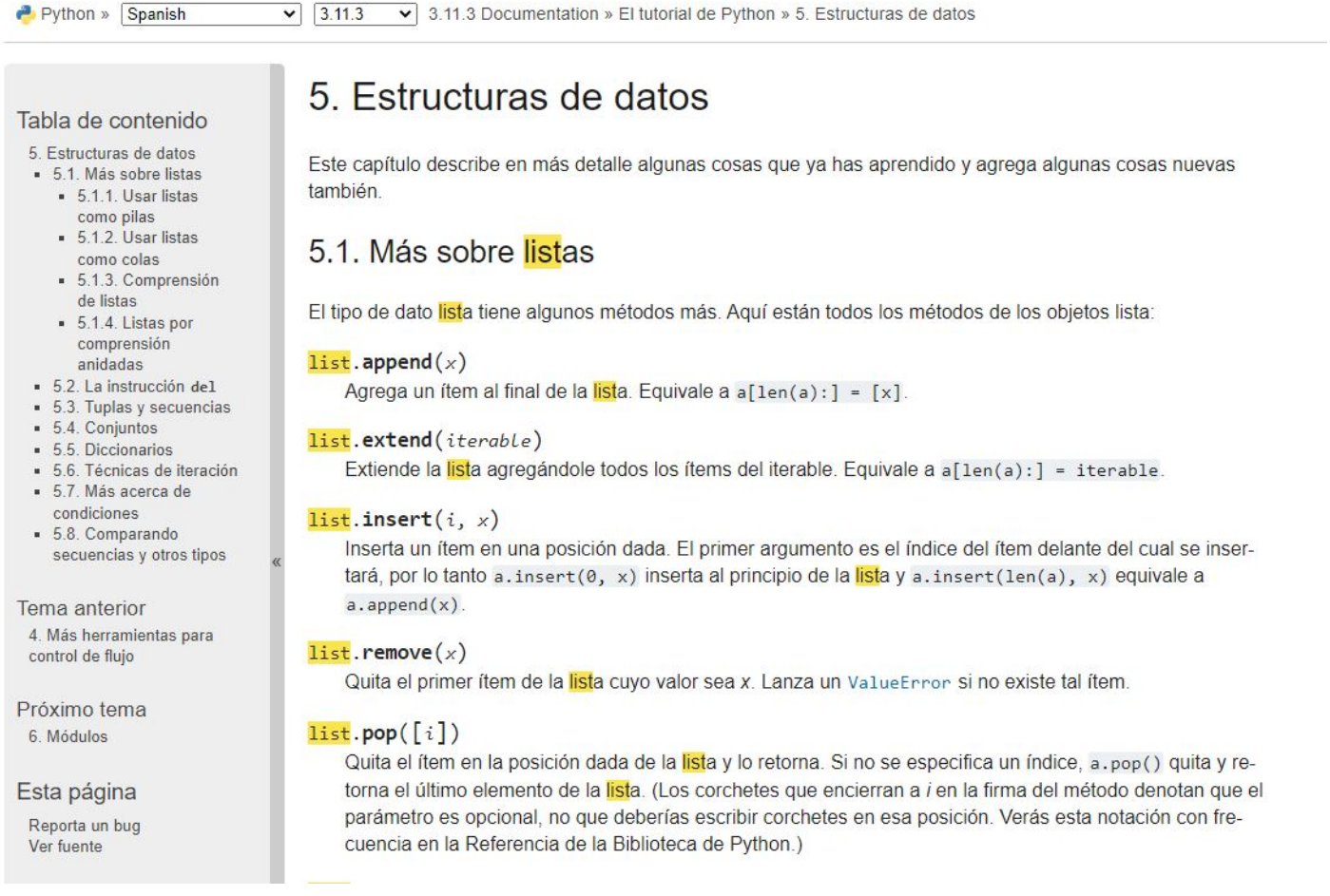
¿Podremos implementar este problema?

problema *traducirTexto*(in nombreArchivo: string, in idiomaOrigen: string, in idiomaDestino: string) : {
  requiere: {El archivo nombreArchivo debe existir.}
  asegura: {Se crea un archivo llamado *idiomaDestino* – *nombreArchivo* cuyo contenido será el resultado de traducir cada una de sus filas}
  asegura: {Si el archivo archivoDestino existia, se borrará todo su contenido anterior}
}

Documentación de Python

Python (como muchos de los otros lenguajes) tiene documentación pública con la descripción del comportamiento de sus distintos tipos de datos.

- https://docs.python.org/es/3/tutorial/datastructures.html?highlight=list



Python » Spanish 3.11.3 3.11.3 Documentation » El tutorial de Python » 5. Estructuras de datos

Tabla de contenido

Tema anterior

4. Más herramientas para control de flujo

Próximo tema

6. Módulos

Esta página

Reporta un bug
Ver fuente

5. Estructuras de datos

Este capítulo describe en más detalle algunas cosas que ya has aprendido y agrega algunas cosas nuevas también.

5.1. Más sobre listas

El tipo de dato lista tiene algunos métodos más. Aquí están todos los métodos de los objetos lista:

list.append(*x*)
Agrega un ítem al final de la lista. Equivale a `a[len(a):] = [x]`.

list.extend(*iterable*)
Extiende la lista agregándole todos los ítems del iterable. Equivale a `a[len(a):] = iterable`.

list.insert(*i*, *x*)
Inserta un ítem en una posición dada. El primer argumento es el índice del ítem delante del cual se insertará, por lo tanto `a.insert(0, x)` inserta al principio de la lista y `a.insert(len(a), x)` equivale a `a.append(x)`.

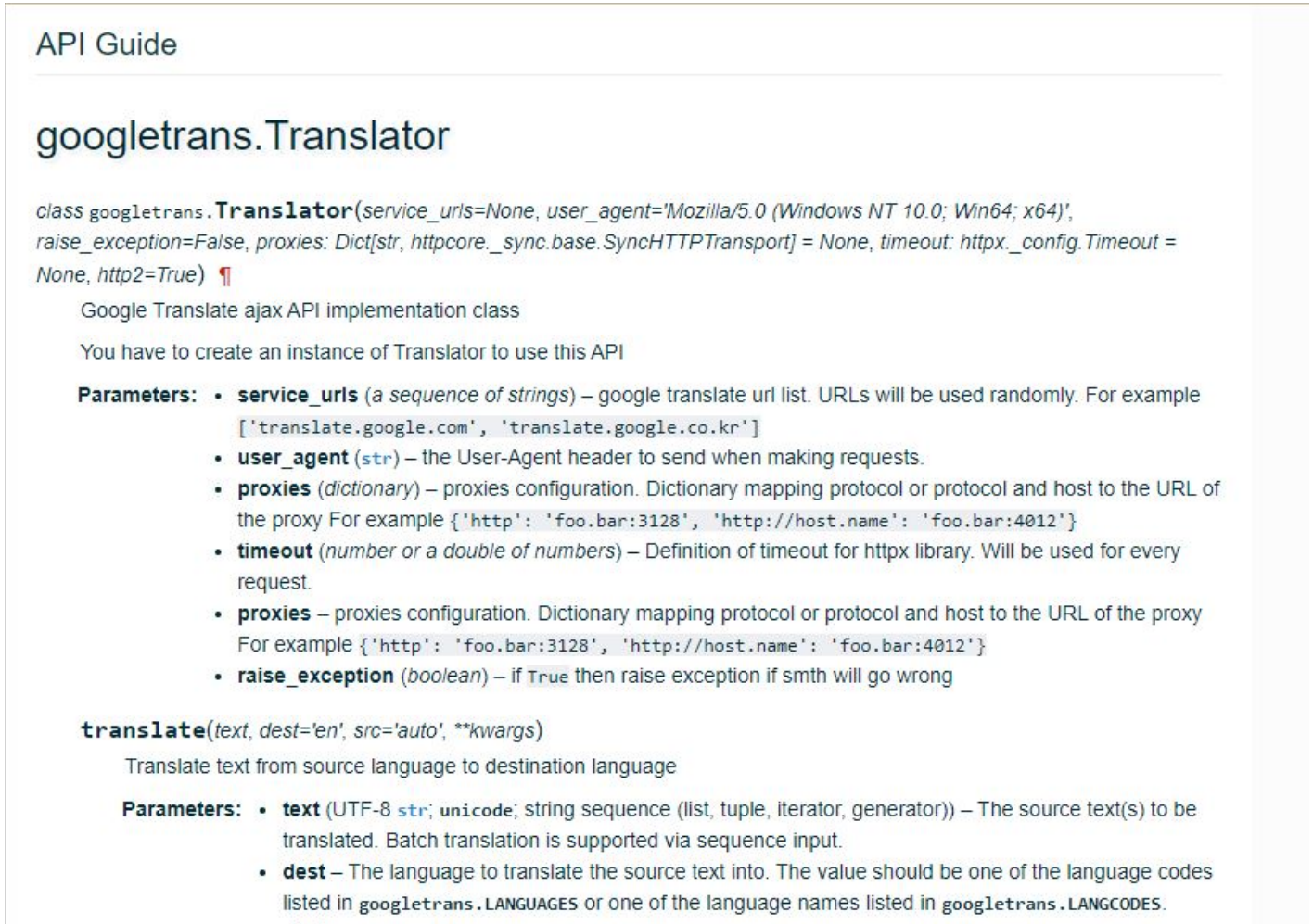
list.remove(*x*)
Quita el primer ítem de la lista cuyo valor sea *x*. Lanza un `ValueError` si no existe tal ítem.

list.pop([*i*])
Quita el ítem en la posición dada de la lista y lo retorna. Si no se especifica un índice, `a.pop()` quita y retorna el último elemento de la lista. (Los corchetes que encierran a *i* en la firma del método denotan que el parámetro es opcional, no que deberías escribir corchetes en esa posición. Verás esta notación con frecuencia en la Referencia de la Biblioteca de Python.)

Documentación de API Google Translate

La API de Google Translate tiene su propia página de especificación

- https://py-googletrans.readthedocs.io/en/latest/



API Guide

googletrans.Translator

class googletrans.**Translator**(*service_urls=None, user_agent='Mozilla/5.0 (Windows NT 10.0; Win64; x64)', raise_exception=False, proxies: Dict[str, httpcore._sync.base.SyncHTTPTransport] = None, timeout: httpx._config.Timeout = None, http2=True*) ¶

Google Translate ajax API implementation class

You have to create an instance of Translator to use this API

Parameters:
- **service_urls** (*a sequence of strings*) – google translate url list. URLs will be used randomly. For example `['translate.google.com', 'translate.google.co.kr']`
- **user_agent** (`str`) – the User-Agent header to send when making requests.
- **proxies** (*dictionary*) – proxies configuration. Dictionary mapping protocol or protocol and host to the URL of the proxy For example `{'http': 'foo.bar:3128', 'http://host.name': 'foo.bar:4012'}`
- **timeout** (*number or a double of numbers*) – Definition of timeout for httpx library. Will be used for every request.
- **proxies** – proxies configuration. Dictionary mapping protocol or protocol and host to the URL of the proxy For example `{'http': 'foo.bar:3128', 'http://host.name': 'foo.bar:4012'}`
- **raise_exception** (*boolean*) – if `True` then raise exception if smth will go wrong

translate(*text, dest='en', src='auto', **kwargs*)

Translate text from source language to destination language

Parameters:
- **text** (UTF-8 `str`; `unicode`; string sequence (list, tuple, iterator, generator)) – The source text(s) to be translated. Batch translation is supported via sequence input.
- **dest** – The language to translate the source text into. The value should be one of the language codes listed in `googletrans.LANGUAGES` or one of the language names listed in `googletrans.LANGCODES`.