Lección 13: Funciones

Índice

Anterior

Definición

Una función es un bloque reutilizable de código o sentencias de programación diseñado para realizar una determinada tarea, con un nombre para poder ser invocado. Para definir o declarar una función, Python proporciona la palabra reservada *def*. El bloque de código únicamente se ejecuta si la función es llamada (no cuando se declara/define).

Librería estándar de Python

La biblioteca estándar de Python es muy amplia, y ofrece una serie de módulos con funciones incorporadas (built-in) y una colección de módulos con otras funciones.

Funciones built-in (incorporadas)

Python proporciona una serie de funciones integradas importantes que podemos usar sin necesidad de proporcionar la definición de la función. Los creadores de Python escribieron un conjunto de funciones para resolver problemas comunes y las incluyeron en Python para que las utilicemos. ### Otros módulos de la librería estándar de python

Son módulos que proveen soluciones estandarizadas para los diversos problemas que pueden ocurrir en el día a día en la programación. Algunos de éstos módulos están diseñados explícitamente para alentar y reforzar la portabilidad de los programas en Python abstrayendo especificidades de las plataformas para lograr APIs neutrales a la plataforma.

- Procesamiento de cadenas de texto: *strings*, *re*, ...
- Tratamiento de datos en bytes: *struct*, *codecs*.
- Otros tipos de datos: *datetime*, *collections*, *enum*, ...
- Módulos numéricos y matemáticos: *numbers*, *math*, *random*, *statistics*, ...
- Acceso a archivos y directorios: *pathlib*, *os.path*, *shutil*, ...
- Persistencia de datos: *pickle*, *sqlite3*, ...
- Compresión y archivo: *zlib*, *gzip*, *bz2*, *zipfile*, *tarfile*, ...
- Formatos de archivo: *csv*, *configparser*, ...
- Criptografía: *hashlib*, *hmac*, *secrets*.
- Servicios genéricos del SO: *so*, *io*, *time*, *argparse*, *logging*, ...
- Ejecución concurrente: *threading*, *subprocess*, ...
- Redes: *socket*, *ssl*, *asyncio*, ...
- Datos en internet: *email*, *json*, *base64*, *mimetypes*, ...
- Proceso de formatos de marcado estructurado: *xml*, *html*, ...

- Protocolos y soporte de internet: *urllib*, *httpserver*, *ftplib*, ...
- Interfaces gráficas de usuario: *tkinter*, ...
- Empaquetado y distribución de software: *distutils*, *venv*, *zipapp*, ...
- Servicios específicos de Windows y Unix
- ...

```
# Ejemplos utilizando la librería estándar de python

import random, string

## Generando contraseñas

length = 20
password = ''
for i in range(length):
    password += random.choice(string.punctuation + string.ascii_lowercase + string.ascii_upp

print(password)

## Generando contraseñas con más aleatoriedad
import secrets
length = 20
password = ''
for i in range(length):
    password += secrets.choice(string.punctuation + string.ascii_lowercase + string.ascii_up

print(password)
```

Declarar y llamar funciones

- Cuando se crea o define el comportamiento de una función, se está **declarando** la función.
- Para utilizar una función hay que **llamarla** o **invocarla**.

Nota: Las funciones se pueden declarar con parámetros o sin ellos.

Función sin parámetros/argumentos

```
# sintaxis en pseudocódigo
# Declarar una función
def nombre_funcion():
    haz esto
    haz esto también
    haz esto también

# Llamar a la función
nombre_funcion()
```

```
# Declarar y llamar funciones

# Declarar una función
def password_gen():
    length = 20
    password = ''
    for i in range(length):
        password += secrets.choice(string.punctuation + string.ascii_lowercase + string.asci
    print(password)


# Llamar una función
password_gen()
```

Funciones devolviendo un valor

Las funciones pueden devolver valores utilizando *return*. Las funciones que no utilizan *return* devuelven el valor *None*.

```
# Funciones devolviendo un valor

# Declarar una función
def password_gen():
    length = 20
    password = ''
    for i in range(length):
        password += secrets.choice(string.punctuation + string.ascii_lowercase + string.asci
    return password


# Llamar una función
mi_contraseña = password_gen()
print(f'Nueva contraseña: {mi_contraseña}')
```

Funciones con parámetros

- A las funciones se les pueden pasar parámetro/argumentos de diferentes tipos (números, cadenas, booleanos, listas, tuplas, diccionarios, conjuntos, otros)

```
#sintaxis pseudocódigo

# Declaración
def nombre_funcion(parametro):
    haz esto
    haz esto también
    haz esto también
```

```
# Llamada
nombre_funcion(argumento)

# Declaración con múltiples argumentos

def nombre_funcion(parametro1, parametro2):
    haz esto
    haz esto también
    haz esto también


# Llamada
nombre_funcion(argumento1, argumento2)
```

```
# Funciones con argumentos

# Declarar una función
def password_gen(length, alphabet):
    password = ''
    for i in range(length):
        password += secrets.choice(alphabet)
    return password

my_alphabet = string.ascii_lowercase + string.ascii_uppercase + string.digits

# Llamar una función mediante argumentos posicionales
mi_contraseña = password_gen(15, my_alphabet)
print(f'Nueva contraseña: {mi_contraseña}')

# Llamar a la función mediante argumentos por clave (por nombre de parámetro)
mi_contraseña = password_gen(alphabet = my_alphabet, length = 15)
print(f'Nueva contraseña: {mi_contraseña}')

# Si se llama la función mezclando argumentos posicionales y nominales, primero deben ir to
```

Con *return* se puede devolver cualquier tipo de dato.

```
# Declarar una función
def es_par(num):
    if num % 2 == 0:
        return True
    else:
        return False
```

```python
# Llamando la función
print(es_par(7))
print(es_par(8))
#print(es_par('Hola'))


# Declarar una función
def es_par(num):
    if type(num) == int or type(num) == float:
        if num % 2 == 0:
            return True
        else:
            return False
    else:
        return None

# Llamando la función
print(es_par(7))
print(es_par(8))
print(es_par('Hola'))

# Declarar una función
def es_par(num):
    try:
        if num % 2 == 0:
            return True
        else:
            return False
    except:
        print('Algo fue mal!')
        return None

# Llamando la función
print(es_par(7))
print(es_par(8))
print(es_par('Hola'))

# Llamar la función con parámetros de menos o de más da error

mi_contraseña = password_gen(15) # Esto da error!!
print(f'Nueva contraseña: {mi_contraseña}')
```

Funciones con parámetros por defecto

Para prevenir lo anterior, se pueden definir parámetros con valores por defecto. Si los parámetros no se pasa, tomarán el valor por defecto.

```
#sintaxis pseudocódigo

# Declaración
def nombre_funcion(parametro = valor):
    haz esto
    haz esto también
    haz esto también


# Llamada
nombre_funcion()
nombre_funcion(argumento)
```

```
# Funciones con argumentos por defecto

# Declarar una función
def password_gen(length, alphabet = string.punctuation + string.ascii_lowercase + string.asc
    password = ''
    for i in range(length):
        password += secrets.choice(alphabet)
    return password


# Llamar una función con un parámetro por defecto
mi_contraseña = password_gen(15)
print(f'Nueva contraseña: {mi_contraseña}')

# Llamar a la función dando una valor al argumento por defecto
my_alphabet = string.punctuation + string.ascii_lowercase
mi_contraseña = password_gen(alphabet = my_alphabet, length = 15)
print(f'Nueva contraseña: {mi_contraseña}')
```

Funciones con un número arbitrario (no definido) de argumentos posicionales - args

- En ocasiones el programador quiere permitir que la función se llame con un número indefinido de argumentos. Esto se consigue añadiendo un * al nombre del parámetro. Ej: **args*
- En el cuerpo de la función los argumentos posicionales serán tratados como en una lista. Ej: Se podría recorrer con *for arg in args*

```
# sintaxis en pseudo código

# Declarar la función
def nombre_funcion(*args):
    haz esto
    haz esto
```

```python
# Llamar a la función
nombre_funcion(param1, param2, param3,..)

# Funciones con un número indeterminado de argumentos

# Definir la función
def sum_all(*nums):
    total = 0       # Dentro de la función nums es una lista con los argumentos pasados a l
    for num in nums:
        total += num
    return total

# Llamar a la función
print(sum_all(5,6,7,8,10,24))
print(sum_all(5,6,7,8))

# Se puede combinar con argumentos definidos y argumentos por defecto

# Definir
def team_list(team, *members):
    print(f'Team name: {team}')
    for member in members:
        print(f'\t- {member}')

# Llamar
team_list('Macacos', 'Marcos', 'Teresa', 'Antonio', 'Pilar')
```

Funciones con un número arbitrario (no definido) de argumentos por clave - kwargs

- De forma similar, pero se consigue añadiendo un ** al nombre del parámetro. Ej: ***kwarg*
- En el cuerpo de la función los argumentos por clave serán tratados como un diccionario. Ej: Se puede recorrer con *for key,value in kwargs.items()*

```python
# Funciones con un número indeterminado de argumentos nominales

# Definir la función
def sum_all(**nums):
    total = 0
    for num in nums.values():
        total += num
    return total

# Llamar a la función
print(sum_all(a=5,b=6,c=7,d=8,e=10,f=24))
```

```
print(sum_all(a=5,b=6,c=7,g=8))

# Se puede combinar con argumentos definidos y argumentos por defecto

# Definir
def team_list(team, **members):
    print(f'Team name: {team}')
    for posicion,member in members.items():
        print(f'\t- {posicion}: {member}')

# Llamar
team_list('Macacos', delantero = 'Marcos', portera = 'Teresa', reserva1 = 'Antonio', reserva
```

Nota: El orden de los parámtros es muy importante, tanto en la definición de la función como en la llamada, y debe ser este : 1. Parámetros estándar (si los hay) 2. Parámetros posicionales indefinidos (*args) 3. Parámetros por clave indefinidos (**kwargs)

Desempaquetar argumentos

- Una lista se puede desempaqueta con * en argumentos posicionales para llamar una función.
- Un diccionario se puede desempaquetar con ** en argumentos por clave para llamar una función.

```
# Desempaquetar lista en argumentos posicionales

# Definicion
def saluda_amigos(*amigos):
    for amigo in amigos:
        print(f'Hola, {amigo}!')

mis_amigos = ['Pepe', 'María', 'Carlos']

saluda_amigos(mis_amigos[0], mis_amigos[1], mis_amigos[2])

saluda_amigos(*mis_amigos)

# Desempaquetar diccionario en argumentos por clave

def team_list(team, **members):
    print(f'Team name: {team}')
    for posicion,member in members.items():
        print(f'\t- {posicion}: {member}')

# Llamar
mi_equipo = {'Delantero': 'Daniel',
```

```
                'Defensa1': 'Pedro',
                'Defensa2': 'Pedro',
                'Portero': 'Lucas'}

team_list('Macacos', **mi_equipo)
```

Documentando funciones - Docstrings

Las cadenas de documentación o docstrings son textos que se escriben entre triples comillas dentro de los programas para documentarlos. Cuando se desarrolla un proyecto donde colaboran varias personas contar con información clara y precisa que facilite la comprensión del código es imprescindible y beneficia a todos los participantes y al propio proyecto.

Las funciones, clases y módulos deben ir convenientemente documentados. La información de las docstrings estará disponible cuando se edite el código y, también, durante la ejecución de los programas.

```
# Documentando funciones

# Declaración
def password_gen(length, alphabet = string.punctuation + string.ascii_lowercase + string.asc
    '''Genera una contraseña segura de longitud length
        Parámetros por clave opcionales:
        alphabet: Cadena con caracteres a utilizar en la contraseña. Por defecto, signos de
    password = ''
    for i in range(length):
        password += secrets.choice(alphabet)
    return password

help(password_gen)
```

Siguiente