Programación en Python
TEMA 7
Diccionarios

Universidad Politécnica de Valencia

2021-2022

Índice

---

1. Diccionarios

Un *diccionario* es como una lista, pero más general. En una lista, los índices de posiciones tienen que ser enteros; en un diccionario, los índices pueden ser (casi) cualquier tipo.

Puedes pensar en un diccionario como una asociación entre un conjunto de índices (que son llamados *claves*) y un conjunto de valores. Cada clave apunta a un valor. La asociación de una clave y un valor es llamada *par clave-valor* o a veces *elemento*.

Como ejemplo, vamos a construir un diccionario que asocia palabras de Inglés a Español, así que todas las claves y los valores son cadenas.

La función `dict` crea un nuevo diccionario sin elementos. Debido a que `dict` es el nombre de una función interna, deberías evitar usarlo como un nombre de variable.

```
>>> eng2sp = dict()
>>> print(eng2sp)
  {}
```

Las llaves, `{}`, representan un diccionario vacío. Para agregar elementos a un diccionario, puedes utilizar corchetes:

```
>>> eng2sp['one'] = 'uno'
```

Esta línea crea un elemento asociando a la clave `'one'` el valor "uno". Si imprimimos el diccionario de nuevo, vamos a ver un par clave-valor con dos puntos entre la clave y el valor:

```
>>> print(eng2sp)
  {'one': 'uno'}
```

Este formato de salida es también un formato de entrada. Por ejemplo, puedes crear un nuevo diccionario con tres elementos. Pero si imprimes `eng2sp`, te vas a sorprender:

```
>>> eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
>>> print(eng2sp)
  {'one': 'uno', 'three': 'tres', 'two': 'dos'}
```

El orden de los pares clave-elemento no es el mismo. De hecho, si tu escribes este mismo ejemplo en tu computadora, podrías obtener un resultado diferente. En general, el orden de los elementos en un diccionario es impredecible.

Pero ese no es un problema porque los elementos de un diccionario nunca son indexados con índices enteros. En vez de eso, utilizas las claves para encontrar los valores correspondientes:

```
>>> print(eng2sp['two'])
  'dos'
```

La clave `'two'` siempre se asocia al valor "dos", así que el orden de los elementos no importa.

Si la clave no está en el diccionario, obtendrás una excepción (`KeyError`):

```
>>> print(eng2sp['four'])
  KeyError: 'four'
```

La función `len` funciona en diccionarios y devuelve el número de pares clave-valor:

```
>>> len(eng2sp)
  3
```

El operador `in` funciona en diccionarios; éste te dice si algo aparece como una *clave* en el diccionario (aparecer como valor no es suficiente).

```
>>> 'one' in eng2sp
  True
>>> 'uno' in eng2sp
  False
```

Para ver si algo aparece como valor en un diccionario, puedes usar el método `values`, el cual retorna los valores como una lista, y después puedes usar el operador `in`:

```
>>> vals = list(eng2sp.values())
>>> 'uno' in vals
  True
```

El operador `in` utiliza diferentes algoritmos para listas y diccionarios. Para listas, utiliza un algoritmo de búsqueda lineal. Conforme la lista se vuelve más grande, el tiempo de búsqueda se vuelve más largo en proporción al tamaño de la lista. Para diccionarios, Python utiliza un algoritmo llamado *tabla hash* (hash table, en inglés) que tiene una propiedad importante: el operador `in` toma la misma cantidad de tiempo sin importar cuántos elementos haya en el diccionario. No vamos a explicar porqué las funciones hash son tan mágicas, pero puedes leer más al respecto en es.wikipedia.org/wiki/Tabla_hash.

Exercise 1:

En Poliformat encuentras una copia del archivo `words.txt` Escribe un programa que lee las palabras de *words.txt* y las almacena como claves en un diccionario. No importa qué valores tenga. Luego puedes utilizar el operador `in` como una forma rápida de revisar si una palabras está en el diccionario.

2. Diccionario como un conjunto de contadores

Supongamos que recibes una cadena y quieres contar cuántas veces aparece cada letra. Hay varias formas en que puedes hacerlo:

1. Puedes crear 26 variables, una por cada letra del alfabeto. Luego puedes recorrer la cadena, y para cada caracter, incrementar el contador correspondiente, probablemente utilizando varios condicionales.
2. Puedes crear una lista con 26 elementos. Después podrías convertir cada caracter en un número (usando la función interna `ord`), usar el número como índice dentro de la lista, e incrementar el contador correspondiente.
3. Puedes crear un diccionario con caracteres como claves y contadores como los valores correspondientes. La primera vez que encuentres un caracter, agregarías un elemento al diccionario. Después de eso incrementarías el valor del elemento existente.

Cada una de esas opciones hace la misma operación computacional, pero cada una de ellas implementa esa operación en forma diferente.

Una *implementación* es una forma de llevar a cabo una operación computacional; algunas implementaciones son mejores que otras. Por ejemplo, una ventaja de la implementación del diccionario es que no tenemos que saber con antelación qué letras aparecen en la cadena y solamente necesitamos espacio para las letras que sí aparecen.

Aquí está un ejemplo de como se vería ese código:

```
palabra = 'brontosaurio'
d = dict()
for c in palabra:
    if c not in d:
        d[c] = 1
    else:
        d[c] = d[c] + 1
print(d)
```

Realmente estamos calculando un *histograma*, el cual es un término estadístico para un conjunto de contadores (o frecuencias).

El bucle `for` recorre la cadena. Cada vez que entramos al bucle, si el caracter `c` no está en el diccionario, creamos un nuevo elemento con la clave `c` y el valor inicial 1 (debido a que hemos visto esta letra solo una vez). Si `c` ya está previamente en el diccionario incrementamos `d[c]`.

Aquí está la salida del programa:

```
{'b': 1, 'r': 2, 'o': 3, 'n': 1, 't': 1, 's': 1, 'a': 1, 'u': 1, 'i': 1}
```

El histograma indica que las letras "a" y "b" aparecen solo una vez; "o" aparece dos, y así sucesivamente.

Los diccionarios tienen un método llamado `get` que toma una clave y un valor por defecto. Si la clave aparece en el diccionario, `get` regresa el valor correspondiente; si no, regresa el valor por defecto. Por ejemplo:

```
>>> cuentas = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
>>> print(cuentas.get('jan', 0))
  100
>>> print(cuentas.get('tim', 0))
  0
```

Podemos usar `get` para escribir nuestro bucle de histograma más conciso. Puesto que el método `get` automáticamente maneja el caso en que una clave no está en el diccionario, podemos reducir cuatro líneas a una y eliminar la sentencia `if`.

```
palabra = 'brontosaurio'
d = dict()
for c in palabra:
    d[c] = d.get(c,0) + 1
print(d)
```

3. Diccionarios y archivos

Uno de los usos más comunes de un diccionario es contar las ocurrencias de palabras en un archivo con algún texto escrito. Vamos comenzando con un archivo de palabras muy simple tomado del texto de *Romeo y Julieta*.

Para el primer conjunto de ejemplos, vamos a usar una versión más corta y más simplificada del texto sin signos de puntuación. Después trabajaremos con el texto de la escena con signos de puntuación incluidos.

```
But soft what light through yonder window breaks
It is the east and Juliet is the sun
Arise fair sun and kill the envious moon
Who is already sick and pale with grief
```

Vamos a escribir un programa de Python para leer a través de las líneas del archivo, dividiendo cada línea en una lista de palabras, y después iterando a través de cada una de las palabras en la línea y contando cada palabra utilizando un diccionario.

Verás que tenemos dos bucles `for`. El bucle externo está leyendo las líneas del archivo y el bucle interno está iterando a través de cada una de las palabras en esa línea en particular. Este es un ejemplo de un patrón llamado *bucles anidados* porque uno de los bucles es el bucle *externo* y el otro bucle es el bucle *interno*.

Como el bucle interno ejecuta todas sus iteraciones cada vez que el bucle externo hace una sola iteración, consideramos que el bucle interno itera "más rápido" y el bucle externo itera más lento.

La combinación de los dos bucles anidados asegura que contemos cada palabra en cada línea del archivo de entrada.

```
fname = input('Ingresa el nombre de archivo: ')
try:
    fhand = open(fname)
except:
    print('El archivo no se puede abrir:', fname)
    exit()

counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        if word not in counts:
            counts[word] = 1
        else:
            counts[word] += 1

print(counts)
```

En nuestra sentencia `else`, utilizamos la alternativa más compacta para incrementar una variable. `counts[word] += 1` es equivalente a `counts[word] = counts[word] + 1`. Cualquiera de los dos métodos puede usarse para cambiar el valor de una variable en cualquier cantidad. Existen alternativas similares para `-=`, `*=`, y `/=`.

Cuando ejecutamos el programa, vemos una salida sin procesar que contiene todos los contadores sin ordenar. El archivo *romeo.txt* está disponible en PoliformaT.

```
>>> %Run
  Ingresa el nombre de archivo: romeo.txt
  {'and': 3, 'envious': 1, 'already': 1, 'fair': 1,
  'is': 3, 'through': 1, 'pale': 1, 'yonder': 1,
  'what': 1, 'sun': 2, 'Who': 1, 'But': 1, 'moon': 1,
  'window': 1, 'sick': 1, 'east': 1, 'breaks': 1,
  'grief': 1, 'with': 1, 'light': 1, 'It': 1, 'Arise': 1,
  'kill': 1, 'the': 3, 'soft': 1, 'Juliet': 1}
```

Es un poco inconveniente ver a través del diccionario para encontrar las palabras más comunes y sus contadores, así que necesitamos agregar un poco más de código para mostrar una salida que nos sirva más.

4. Bucles y diccionarios

Si utilizas un diccionario como una secuencia para un bucle `for`, esta recorre las claves del diccionario. Este bucle imprime cada clave y su valor correspondiente:

```
contadores = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
for clave in contadores:
    print(clave, contadores[clave])
```

Aquí está lo que muestra de salida:

```
jan 100
chuck 1
annie 42
```

De nuevo, las claves no están en ningún orden en particular.

Podemos utilizar un bucle `for` de esta manera con diccionarios para implementar diferentes cosas. Por ejemplo, si queremos encontrar todas las entradas en un diccionario con valor mayor a diez, podemos escribir el siguiente código:

```
contadores = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
for clave in contadores:
    if contadores[clave] > 10 :
        print(clave, contadores[clave])
```

El bucle `for` itera a través de las *claves* del diccionario, así que debemos utilizar el operador índice para obtener el *valor* correspondiente a cada clave. Aquí está la salida del programa:

```
jan 100
annie 42
```

Vemos solamente las entradas que tienen un valor mayor a 10.

Si quieres imprimir las claves en orden alfabético, primero haces una lista de las claves en el diccionario utilizando el método `keys` disponible en los objetos de diccionario, y después ordenar esa lista e iterar a través de la lista ordenada, buscando cada clave e imprimiendo pares clave-valor ordenados, tal como se muestra a continuación:

```
contadores = { 'chuck' : 1 , 'annie' : 42, 'jan': 100}
lst = list(contadores.keys())
print(lst)
lst.sort()
for clave in lst:
    print(clave, contadores[clave])
```

Así se muestra la salida:

```
['jan', 'chuck', 'annie']
annie 42
chuck 1
jan 100
```

Primero se ve la lista de claves sin ordenar como la obtuvimos del método `keys`. Después vemos los pares clave-valor en orden desde el bucle `for`.

5. Análisis avanzado de texto

En el ejemplo anterior utilizando el archivo *romeo.txt*, hicimos el archivo tan simple como fue posible removiendo los signos de puntuación a mano. El text real tiene muchos signos de puntuación, como:

```
But, soft! what light through yonder window breaks?
It is the east, and Juliet is the sun.
Arise, fair sun, and kill the envious moon,
Who is already sick and pale with grief,
```

Puesto que la función `split` en Python busca espacios y trata las palabras como piezas separadas por esos espacios, trataríamos a las palabras "soft!" y "soft" como *diferentes* palabras y crearíamos una entrada independiente para cada palabra en el diccionario. Además, como el archivo tiene letras mayúsculas, trataríamos "who" y "Who" como diferentes palabras con diferentes contadores.

Podemos resolver ambos problemas utilizando los siguientes métodos de cadenas:

1. `lower`: reemplaza todos los mayúsculas con su minusculas correspondientes:

```
>>> "HeLlOo!".lower()
'helloo!'
```

2. `punctuation`: devuelve los caracteres que Python considera como "signos de puntuación":

```
>>> import string
>>> string.punctuation
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

3. `translate`, el más sutil de los métodos. Aquí esta la documentación para `translate`:

`line.translate(str.maketrans(fromstr, tostr, deletestr))`

Reemplaza los caracteres en `fromstr` *con el caracter en la misma posición en* `tostr` *y elimina todos los caracteres que están en* `deletestr`*. Los parámetros* `fromstr` *y* `tostr` *pueden ser cadenas vacías y el parámetro* `deletestr` *es opcional.*

No vamos a especificar el valor de `tostr` pero vamos a utilizar el parámetro `deletestr` para eliminar todos los signos de puntuación.

Hacemos las siguientes modificaciones a nuestro programa:

```
import string

fname = input('Ingresa el nombre de archivo: ')
try:
    fhand = open(fname)
except:
    print('El archivo no se puede abrir:', fname)
    exit()

counts = dict()
for line in fhand:
    line = line.rstrip()
    line = line.translate(line.maketrans('', '', string.punctuation))
    line = line.lower()
    words = line.split()
    for word in words:
        if word not in counts:
```

```
            counts[word] = 1
        else:
            counts[word] += 1

print(counts)
```

Parte de aprender el "Arte de Python" o "Pensamiento Pythónico" es entender que Python muchas veces tiene funciones internas para muchos problemas de análisis de datos comunes. A través del tiempo, verás suficientes códigos de ejemplo y leerás lo suficiente en la documentación para saber dónde buscar si alguien escribió algo que haga tu trabajo más fácil.

Lo siguiente es una versión reducida de la salida:

```
Ingresa el nombre de archivo: romeo-full.txt
{'swearst': 1, 'all': 6, 'afeard': 1, 'leave': 2, 'these': 2,
'kinsmen': 2, 'what': 11, 'thinkst': 1, 'love': 24, 'cloak': 1,
a': 24, 'orchard': 2, 'light': 5, 'lovers': 2, 'romeo': 40,
'maiden': 1, 'whiteupturned': 1, 'juliet': 32, 'gentleman': 1,
'it': 22, 'leans': 1, 'canst': 1, 'having': 1, ...}
```

Interpretar los datos a través de esta salida es aún difícil, y podemos utilizar Python para darnos exactamente lo que estamos buscando, pero para que sea así, necesitamos aprender acerca de las *tuplas* en Python. Vamos a retomar este ejemplo una vez que aprendamos sobre tuplas.

6. Depuración

Conforme trabajes con conjuntos de datos más grandes puede ser complicado depurar imprimiendo y revisando los datos a mano. Aquí hay algunas sugerencias para depurar grandes conjuntos de datos:

Reducir la entrada Si es posible, trata de reducir el tamaño del conjunto de datos. Por ejemplo, si el programa lee un archivo de texto, comienza solamente con las primeras 10 líneas, o con el ejemplo más pequeño que puedas encontrar. Puedes ya sea editar los archivos directamente, o (mejor) modificar el programa para que solamente lea las primeras `n` número de líneas.

Si hay un error, puedes reducir `n` al valor más pequeño que produce el error, y después incrementarlo gradualmente conforme vayas encontrando y corrigiendo errores.

Revisar extractos y tipos En lugar de imprimir y revisar el conjunto de datos completo, considera imprimir extractos de los datos: por ejemplo, el número de elementos en un diccionario o el total de una lista de números.

Una causa común de errores en tiempo de ejecución es un valor que no es el tipo correcto. Para depurar este tipo de error, generalmente es suficiente con imprimir el tipo de un valor.

Escribe auto-verificaciones Algunas veces puedes escribir código para revisar errores automáticamente. Por ejemplo, si estás calculando el promedio de una lista de números, podrías verificar que el resultado no sea más grande que el elemento más grande de la lista o que sea menor que el elemento más pequeño de la lista. Esto es llamado "prueba de sanidad" porque detecta resultados que son "completamente ilógicos".

Otro tipo de prueba compara los resultados de dos diferentes cálculos para ver si son consistentes. Esto es conocido como "prueba de consistencia".

Imprimir una salida ordenada Dar un formato a los mensajes de depuración puede facilitar encontrar un error.

De nuevo, el tiempo que inviertas haciendo una buena estructura puede reducir el tiempo que inviertas en depurar.

7. Glosario

bucles anidados Cuando hay uno o más bucles "dentro" de otro bucle. Los bucles internos terminan de ejecutar cada vez que el bucle externo ejecuta una vez.
búsqueda Una operación de diccionario que toma una clave y encuentra su valor correspondiente.
clave Un objeto que aparece en un diccionario como la primera parte de un par clave-valor.
diccionario Una asociación de un conjunto de claves a sus valores correspondientes.
elemento Otro nombre para un par clave-valor.
función hash Una función utilizada por una tabla hash para calcular la localización de una clave.
histograma Un set de contadores.
implementación Una forma de llevar a cabo un cálculo.
par clave-valor La representación de una asociación de una clave a un valor.
tabla hash El algoritmo utilizado para implementar diccionarios en Python.
valor Un objeto que aparece en un diccionario como la segunda parte de un par clave-valor. Esta definición es más específica que nuestro uso previo de la palabra "valor".

Ejercicios

Exercise 2:

El alfabeto de la *International Civil Aviation Organization* (ICAO) asigna un código de palabras a las letras del alfabeto inglés (alfa para la A, bravo para la B, etc.), de forma que las combinaciones de letras críticas se pueden pronunciar y pueden ser entendidas por aquellos que transmiten y reciben mensajes de voz por radio o teléfono, independientemente de su idioma nativo. Imagina que tenemos el siguiente diccionario en Python asignado en la variable `dic_ICAO`:

```
dic_ICAO = {
    'a':'alfa',
    'b':'bravo',
    'c':'charlie',
    'd':'delta',
    'e':'echo',
    'f':'foxtrot',
    'g':'golf',
    'h':'hotel',
    'i':'india',
    'j':'juliett',
    'k':'kilo',
    'l':'lima',
    'm':'mike',
    'n':'november',
    'o':'oscar',
    'p':'papa',
    'q':'quebec',
    'r':'romeo',
    's':'sierra',
    't':'tango',
    'u':'uniform',
    'v':'victor',
    'w':'whiskey',
    'x':'x-ray',
    'y':'yankee',
    'z':'zulu'
    }
```

Diseña una función que reciba un texto y sea capaz de traducirlo en la lista de palabras ICAO. Puedes usar la siguiente plantilla (template) que tambien esta en PoliformaT.

```python
def traducir_a_ICAO (texto):

    """
    Esta función traduce un texto al alfabeto ICAO definido en
    el diccionario dic_ICAO.

    Entradas: Un texto a traducir
    Salidas:  Un texto traducido a ICAO
    Restricciones: solo funciona con texto de tipo str.
    """

#    <TU SOLUCIÓN AQUI>


@pytest.mark.parametrize("testcase, entrada, salida_esperada",[

#    <TUS CASOS DE TEST AQUI>

])

def test_traducir_a_ICAO(testcase, entrada, salida_esperada):
    assert traducir_a_ICAO(entrada) == salida_esperada,\
           "caso {0}".format(testcase)
```

Exercise 3:

Considera la siguiente función en Python:

```python
def traducir(dic, source):
    result = dic[source[0]]
    for i in range(1, len(source)):
        result = result + dic[source[i]]
    return result
```

Imagina tambien que tenemos los siguientes diccionarios definidos:

```python
dic_ej1 = {
    1: 'one',
    2: 'two',
    3: 'three'
    }

dic_ej2 = {
    1: [1],
    2: [2],
    3: [3]
    }

dic_ej3 = {
    1: 1,
    2: 2,
    3: 3
    }

dic_ej4 = {
    '1': 1,
    '2': 2,
```

```
    '3': 3
    }
```

¿Qué respuesta da Python para las siguientes instrucciones?

```
1. >>> traducir (dic_ej1, (1,2,1))
2. >>> traducir (dic_ej1, [1,2,1])
3. >>> traducir (dic_ej2, [1,2,1])
4. >>> traducir (dic_ej2, 123)
5. >>> traducir (dic_ej3, [1,2,1])
6. >>> traducir (dic_ej4, [1,2,1])
7. >>> traducir (dic_ej4, "123")
8. >>> traducir (dic_ej4, ['1', '2', '3'])
```

Exercise 4:

Escribe la documentación para la función `traducir`, que hace la función para que queda claro para que parámetros funciona y para que parámetros lanza excepciones y cuales.

Exercise 5:

Los siguientes tests son para testear algunos de los ejemplos de más arriba (i.e. 1 hasta 8).

```
@pytest.mark.parametrize("testcase, input1, input2, output",[
(a, dic_ej2, [1,2,1], [1,2,1]),
(b, dic_ej1, (1,2,1), 'onetwoone'),
(c, dic_ej1, [1,2,1], 'onetwoone'),
(d, dic_ej3, [1,2,1], 4),
(e, dic_ej4, ['1', '2', '3'], 6)
(f, dic_ej4, "123", 6),
])

def test_traducir(testcase, input1, input2, output):
    assert traducir(input1, input2) == output,\
           "caso {0}".format(testcase)
```

¿Con qué números correspondan `a`, `b`, `c`, `d`, `e`, `f`?

Exercise 6:

Hemos visto que el metodo `traducir` lanza un `TypeError` cuando el `source` tiene un tipo que no es *subscriptable*. Tambien vimos un `KeyError` cuando el clave con que intentamos acceder al diccionario no coincide con el tipo del diccionario.

Para escribir `pytest` que verifican que salen estas excepciones se puede escribir los siguientes tests:

```
@pytest.mark.parametrize("testcase, input1, input2, output",[
(g, dic_ej2, 123, pytest.raises(TypeError)),
(h, dic_ej4, [1,2,1], pytest.raises(KeyError))
])

def test_traducir_exceptions(testcase, input1, input2, output):
    with output:
        traducir(input1, input2)
```

¿Con qué números correspondan `g`, `h`?

Exercise 7:

Vamos a implementar un programa para controlar el stock de una frutería. Para ello, tenemos dos diccionarios:

```
precio = {
    "platano": 3,
    "manzana": 2,
    "naranja": 2.5,
    "pera": 3.5
    }

stock = {
    "platano": 20,
    "manzana": 30,
    "naranja": 15,
    "pera": 30
    }
```

Empezamos con una función `valor_del_stock` que calcula, como indica su nombre, el valor que tiene el stock.

¿Con qué valores vas a ejecutar tu función para testearlo bien?

Exercise 8:

Ahora implementa una función `venta` que simula una venta. Para ello, la función recibe como parámetros el nombre de la fruta que se vende y la cantidad. La función deberá descontar la cantidad vendida del stock. Si no hay stock, se mostrará un mensaje de error.

¿Con qué valores vas a ejecutar tu función para testearlo bien?

Exercise 9:

Ahora implementa una función `compra` que simula una compra. Para ello, la función recibe como parametros el nombre de la fruta que se ha comprado y la cantidad. La función deberá actualizar la cantidad comprado al stock. Si este producto no ha aparecido antes en el stock (i.e. melones), hay que añadirlo.

¿Con qué valores vas a ejecutar tu función para testearlo bien?

Exercise 10:

Una anagrama es una palabra o frase que se forma al reorganizar las letras de otra palabra o frase. Por ejemplo, ”amor” es un anagrama de ”roma”, y ”frase” es una anagrama de ”fresa”. Escriba una función `is_anagram (word1, word2)` que verifique si las dos palabras son anagramas entre sí. Si es así, la función debería devolver True, y False de lo contrario. La función no debe distinguir entre letras mayúsculas y minúsculas. Puedes usar las siguientes ejemplos para testear la función con pytest:

testcase número	input	output esperado
1	(frase, fresa)	True
2	(panel, nepal)	True
3	(Alumno, Molan)	False
4	(Hola, Caracola)	False
5	(Rectificable, certificable)	True

Exercise 11:

Imagine un pequeño albergue con cuatro habitaciones de cuatro camas (con los números elegidos arbitrariamente 101, 102, 201 y 202). Vamos a escribir un programa Python para el personal del albergue para ayudarlos a realizar un seguimiento de la ocupación de la habitación y registrar la entrada y salida de los huéspedes.

El código Python para la interacción del usuario ya existe:

```
room_occupancy = {101:[], 102:[], 201:[], 202:[]}
while True:
    print("These are your options:")
    print("1 — View current room occupancy.")
    print("2 — Check guest in.")
    print("3 — Check guest out.")
    print("4 — Exit program.")
    choice = input("Please choose what you want to do: ")
    if choice == "1":
        print_occupancy(room_occupancy)
    elif choice == "2":
        guest = input("Enter name of guest: ")
        room = int(input("Enter room number: "))
        check_in(room_occupancy, guest, room)
    elif choice == "3":
        guest = input("Enter name of guest: ")
        room = int(input("Enter room number: "))
        check_out(room_occupancy, guest, room)
    elif choice == "4":
        print("Goodbye!")
        break
    else:
        print("Invalid input, try again.")
```

Vamos a escribir un par de funciones más:

- `print_occupancy` simplemente debe imprimir una lista de todas las habitaciones y los huéspedes que se registran actualmente.
- `check_in` debería agregar un invitado a una habitación. Si se proporciona un número de habitación no existente o si la habitación elegida ya está llena, se debe imprimir el mensaje correspondiente. Puede haber dos invitados con el mismo nombre en una habitación
- `check_out` debe eliminar a un invitado de una habitación. Si se pasa un número de habitación o nombre de invitado incorrecto, se debe imprimir el mensaje correspondiente