Status	Finished
Started	Monday, 2 December 2024, 6:04 PM
Completed	Monday, 2 December 2024, 6:07 PM
Duration	2 mins 23 secs
Marks	7.00/7.00
Grade	**10.00** out of 10.00 (**100**%)

Information

Diccionarios

Los diccionarios, conocidos en algunos lenguajes como *arrays asociativos*, son colecciones de datos, como las secuencias, pero, a diferencia de estas, no se indexan usando un rango de números (0, 1, 2, ···) sino que usan una **clave**, que pueden ser de cualquier tipo inmutable (típicamente, strings o números). Un diccionario (tipo **`dict`**) es una colección **no ordenada** de parejas clave-valor.

Un diccionario inicialmente vacío se crea con unas llaves vacías:

```
d = {}
type(d)
<class 'dict'>
len(d)
0
```

También se puede crear un diccionario vacío usando sin argumentos el constructor **`dict()`**:

```
d = dict()
```

Un diccionario de Python se puede crear no vacío inicializándolo con uno o más elementos con la forma

clave **:** *valor*

separados por comas. El siguiente es un (mini) diccionario inglés-español, donde tanto claves como valores son strings, las primeras las palabras en inglés y los segundos su traducción al español:

```
d = {'mother': 'madre', 'father': 'padre', 'house': 'casa', 'car': 'coche'}
```

Question 1

Complete

Mark 1.00 out of 1.00

Empareje cada uno de los siguientes diccionarios con sus tipos de claves y valores.

`b = {'Peter': [100, 75, 99], 'Martin': [77, 75, 77]}`	Claves str, valores list
`c = {728: 'Peter Carrot', 27: 'John Cabbage', 140: 'Andrea Plum'}`	Claves int, valores str
`a = {'and': 10, 'you': 12, 'meeting': 2}`	Claves str, valores int

Information

Otras formas de inicializar un diccionario

Al constructor *dict()* se le puede pasar una secuencia de pares clave:valor. El siguiente ejemplo inicializa un (mini) diccionario inglés-eslovaco:

```
my_list = [('mother', 'matka' ), ('father', 'otec'), ('house', 'dom'), ('car', 'auto')]
d = dict(my_list)
print(d) # {'mother': 'matka', 'father': 'otec', 'house': 'dom', 'car': 'auto'}
```

También se puede usar comprehension para crear diccionarios:

```
d = {x: x**2 for x in (2, 4, 6)}
print(d) # {2:4, 4:16, 6:36}
```

Cuando las claves son strings, se pueden usar como "parámetros por nombre" de *dict()*:

```
d = dict(Michael=77, Peter=66, Patrick=82)
print(d)  # {'Michael':77, 'Peter':66, 'Patrick':82}
```

Question 2

Complete

Mark 1.00 out of 1.00

Empareje cada sentencia con el diccionario resultante.

`d = {x:random.randrange(x) for x in range(1,5)}`	{1: 0, 2: 0, 3: 2, 4: 3}
`d = dict(A=1, B=1.5, C=2, D=2.5, E=3, FX=4)`	{'A': 1, 'B': 1.5, 'C': 2, 'D': 2.5, 'E': 3, 'FX': 4}
`d = dict([(2, 'bin'), (8, 'oct'), (16, 'hex')])`	{2: 'bin', 8: 'oct', 16: 'hex'}

Information

Operaciones en diccionarios

Las principales operaciones que se pueden realizar en un diccionario son:

- Almacenar un valor asociado a una clave.
- Obtener el valor asociado a una clave.
- Borrar un par clave:valor.

```
d = {'mother': 'madre', 'father': 'padre'}
d['child'] = 'hijo'
print(d) # {'mother': 'madre', 'father': 'padre', 'child': 'hijo'}
```

En el ejemplo anterior, la instrucción d['child'] = 'hijo' añade un nuevo par clave:valor al diccionario d. Si se usa una clave ya existente, el nuevo valor sustituye al almacenado previamente:

```
d['child'] = 'hija'
print(d) # {'mother': 'madre', 'father': 'padre', 'child': 'hija'}
```

Para acceder a un valor usamos su clave:

```
print(d['mother']) # madre
```

Es un error intentar acceder al diccionario usando una clave que no existe:

```
v = d['dog']
Traceback (most recent call last): File "<input>", line 7, in <module>
KeyError: 'dog'
```

Podemos prevenirlo usando el operador ***in***, que nos permite saber si una clave existe:

```
if 'dog' in d:
   print(d['dog'])
else:
   print('La clave "dog" no existe en el diccionario')
```

Question 3

Complete

Mark 1.00 out of 1.00

Dado el diccionario resultante del siguiente código:

```
t = dict(A=10, B=12, C=8)
t['D']= 9
t['FX'] = 2
```

Empareje cada intrucción con su resultado.

print(t)	{'FX': 2, 'B': 12, 'C': 8, 'A': 10, 'D': 9}
print(len(t))	5
print(t['C'])	8
print(t['X'])	KeyError
print('FX' in t)	True

Information

Eliminar una pareja clave:valor de un diccionario

A veces, necesitamos eliminar una pareja clave:valor de un diccionario. Podemos hacerlo con el método ***pop()***.

```
d = {'mother': 'madre', 'father': 'padre', 'child': 'hijo'}
value = d.pop('mother')
print(d) # {'father': 'padre', 'child': 'hijo'}
```

El método *pop*() elimina el par clave:valor asociado a la clave pasada como parámetro y devuelve el correspondiente valor. Si no nos interesa quedarnos con el valor, podemos llamar al método *pop()* sin asignar su resultado a ninguna variable:

```
d.pop('mother')
```

La instrucción ***del*** podemos usarla también con diccionarios (obsérvese cómo ha de escribirse):

```
d = {'mother': 'madre', 'father': 'padre', 'child': 'hijo'}
del d['mother']
print(d) # {'father': 'padre', 'child': 'hijo'}
```

En ambos casos, se produce un error (*KeyError*) si la clave en cuestión no existe en el diccionario.

Para vaciar un diccionario se usa el método ***clear()***:

```
d = {'mother': 'madre', 'father': 'padre', 'child': 'hijo'}
d.clear()
print(d) # {}
```

Question 4

Complete

Mark 1.00 out of 1.00

Marque las afirmaciones que sean verdaderas.

Select one or more:

- [x] Cuando se usa la instrucción **del** con un diccionario puede producirse un "KeyError".
- [ ] El método **pop()** devuelve un valor cualquiera de un diccionario.
- [x] El método **clear()** vacía el diccionario.

Information

El método *get()*.

A veces, es mejor usar el método *get()* para recuperar el valor asociado a una clave:

```
capitals = {'Slovakia': 'Bratislava', 'Austria': 'Vienna'}
print(capitals['Slovakia'])     # Bratislava
print(capitals.get('Slovakia')) # Bratislava
```

Supongamos que la clave no está en el diccionario:

```
capitals = {'Slovakia': 'Bratislava', 'Austria': 'Vienna'}
print(capitals.get('Hungary')) # None
print(capitals['Hungary'])
Traceback (most recent call last):
  File "my_program.py", line 2, in <module>
    print(capitals['Hungary'])
KeyError: 'Hungary'
```

El método *get()* no produce un error cuando la clave no se encuentra en el diccionario; en su lugar se devuelve el valor *None*, a menos que se especifique un valor diferente añadiendo un segundo parámetro en la llamada:

```
print(capitals.get('Hungary', '?')) # ?
```

Question 5

Complete

Mark 1.00 out of 1.00

Dado el siguiente código:

```
t = {'A': 10, 'C': 8, 'D': 9, 'E': 4, 'FX': 2}
x = input('next student's result:')
t[x] = t.get(x, 0) + 1
print(t)
```

Empareje cada entrada de usuario con el resultado que se muestra (recuérdese que un diccionario es una **colección no ordenada** de parejas clave:valor)

El usuario entra el caracter 'B' {'A': 10, 'C': 8, 'D': 9, 'E': 4, 'FX': 2, 'B': 1}

El usuario entra el caracter 'C' {'FX': 2, 'E': 4, 'C': 9, 'A': 10, 'D': 9}

Information

Iterando en diccionarios

Aparte de obtener un valor usando una clave, podemos obtener todas las claves, todos los valores, o todos los pares *clave:valor* de un diccionario:

```
t = {'A': 10, 'B': 9, 'C': 9, 'D': 9, 'E': 4, 'FX': 2}
print(t.keys())
print(t.values())
print(t.items())
```

Resultado:

```
dict_keys(['A', 'B', 'C', 'D', 'E', 'FX'])
dict_values([10, 9, 9, 9, 4, 2])
dict_items([('A', 10), ('B', 9), ('C', 9), ('D', 9), ('E', 4), ('FX', 2)])
```

Los cuales son todos ellos objetos iterables:

```
for k in t.keys():
    print(k, end=' ')      # A B C D E FX

for v in t.values():
    print(v, end=' ')      # 10 9 9 9 4 2

for k, v in t.items():     # A 10
    print(k, v)            # B 9
                           # C 9
                           # D 9
                           # E 4
                           # FX 2
```

También se puede recorrer un diccionario simplemente con:

```
for k in t:
    print(k, t[k])
```

Los objetos resultantes de llamar a los métodos *keys()*, *values()* o *items()* pueden convertirse a su vez en listas, p.e. la expresión

```
list(d.keys())
```

devuelve una lista con todas las claves del diccionario d, en un **orden** que podemos considerar **arbitrario** (recuérdese que un diccionario es una colección **no ordenada** de parejas clave:valor). Si se necesita tener las claves ordenadas, debe usarse en su lugar la expresión

```
sorted(d.keys())
```

Question 6

Complete

Mark 1.00 out of 1.00

Dado el siguiente diccionario:

```
d = {"and": 10, "for": 2, "if": 8}
```

complete el siguiente bucle para mostrar todas las parejas clave:valor que contiene:

```
for k, v in  d.items()  :
    print(k, v)
```

Information

Diccionarios como valores de diccionarios

Un diccionario puede ser un valor en otro diccionario:

```
s1 = {'name': 'Paul Carrot', 'address': {'Street': 'Long', 'Number': 13, 'City': 'Nitra'}}
s2 = {'name': 'Peter Pier', 'address': {'Street': 'Short', 'Number': 21, 'City': 'Nitra'}}
s3 = {'name': 'Patrick Nut', 'address': {'Street': 'Deep', 'Number': 77, 'City': 'Nitra'}}
students = [s1, s2, s3]
```

En el ejemplo anterior se crea una lista de 3 diccionarios.

El elemento students[0] es un diccionario con dos parejas clave-valor:

```
print(student[0]['name']) # Paul Carrot
print(student[0]['address']) # {'Street': 'Long', 'Number': 13, 'City': 'Nitra'}
```

El valor asociado a la clave 'address' es, a su vez, un diccionario:

```
print(student[0]['address']['Street']) # Long
print(student[0]['address']['Number']) # 13
print(student[0]['address']['City'])   # Nitra
```

Question 7

Complete

Mark 1.00 out of 1.00

Dado el siguiente diccionario:

```
person = {'name': 'Mathew', 'birthday': {'day': 12, 'month': 3, 'year': 2000}}
```

Completar para mostrar el año de nacimiento de la persona:

```
print(person['birthday'] ['year']
```