ESTRUCTURA DE DATOS
Diccionarios

Un diccionario es un dato especial de *Python* que almacena una serie de valores utilizando otros como referencia para su acceso y almacenamiento. Cada elemento de un diccionario es un par clave-valor (*key-value*) donde el primero debe ser único y será usado para acceder al valor que contiene. A diferencia de las tuplas y las listas, los diccionarios no cuentan con un orden específico, siendo el intérprete de *Python* el encargado de decidir el orden del almacenamiento. Sin embargo, un diccionario es iterable, mutable y representa una colección de objetos que pueden ser de diferentes tipos (*Fernández-Montoro, 2014*).

Para declarar un diccionario se utilizan las llaves ({}) entre los que se encuentran los pares *key-value* separados por comas. La *key* de cada elemento aparece separada del correspondiente *value* por el carácter *dos puntos (:).* Alternativamente, podemos hacer uso de la función *dict()* para crear un diccionario.

A diferencia de las listas y tuplas, el acceso al valor de un elemento no es por índice sino por la clave (*key*). Los métodos principales que nos permiten iterar sobre un diccionario son: *items()*, *values()* y *keys()*. El primero da acceso tanto a claves como a valores, el segundo se encarga de devolver los valores y el último, las claves del diccionario.

```
"""Programa que crea diccionarios por comprension para raiz cuadrada, seno, coseno y tangente
de un angulo dado en grados sexagesimales y convertido a radianes"""
import math as m
numeros=list(range(0,91))
TablaRC={k:m.sqrt(k*m.pi/180) for k in numeros}
TablaSin={k:m.sin(k*m.pi/180) for k in numeros}
TablaCos={k:m.cos(k*m.pi/180) for k in numeros}
TablaTan={k:m.tan(k*m.pi/180) for k in numeros}
print("Búsqueda de valores Sqrt, Sin, Cos, Tan, Radian de 0 a 90 grados")
print()
valor=eval(input("Introduzca un ángulo (sexagesimales): "))
print("Angulo--Raiz cuadrada(radian)--Seno(radian)--Coseno(radian)--Tan(radian)--Radian")
print(valor,"--",TablaRC[valor],"--",TablaSin[valor],"--",TablaCos[valor],"--",TablaTan[valor],"--",valor*m.pi/180)
```

```
Búsqueda de valores Sqrt, Sin, Cos, Tan, Radian de 0 a 90 grados

Introduzca un ángulo (sexagesimales): 45
Angulo--Raiz cuadrada(radian)--Seno(radian)--Coseno(radian)--Tan(radian)--Radian
45 -- 0.8862269254527579 -- 0.7071067811865475 -- 0.7071067811865476 -- 0.9999999999999999 -- 0.7853981633974483
>>>
```

El siguiente programa toma el *Zen de Python* creado por *Tim Peters*, lo muestra en inglés de la biblioteca importada *this.py* y permite traducir por línea (19 líneas) o en su totalidad. Con la tecla <*F*> se termina el programa.

```
"""Zen de Python"""
Zen={1:"Bonito es mejor que feo",2:"Explícito es mejor que implícito",3:"Simple es mejor que complejo",
     4:"Complejo es mejor que complicado",5:"Sencillo es mejor que anidado",6:"Disperso es mejor que denso",
     7:"La legibilidad cuenta",8:"Los casos especiales no son lo suficientemente especiales para romper las reglas",
     9:"La practicidad bate a la pureza",10:"Los errores no deben pasar inadvertidos",11:"A menos que sean explícitamente silenciados",
     12:"En caso de ambigüedad, rechazar la tentación de adivinar",13:"Debe haber una -preferiblemente única- forma obvia de hacerlo.",
     14:"Aunque es única forma no sea obvia en un primer momento, a no ser que seas holandés", 15:"Ahora es mejor que nunca",
     16:"Aunque nunca es mejor que 'ahora' mismo",17:"Si la implementación es difícil de explicar, es una mala idea",
     18:"Si la implementación es fácil de explicar, debe ser una buena idea",
     19:"Los espacios de nombres son una idea genial-¡hay que hacer más de esos!"}
import this
print()
n="F"
while n!="F" or n!="f":
    n=input("¿Cuál línea quiere traducir (1-19), T(odas), F(in): ")
    if n in list(str(i) for i in range(1,20)):
        num=eval(n)
        print()
        if num>=1 and num<=19:
            print(Zen[num])
            print()
        else:
            print("No existe tal línea")
    else:
        print()
        if n=="T" or n=="t":
            for i in range(1,len(Zen)):
                print(Zen[i])
        print()
        if n=="F" or n=="f":
            break
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

¿Cuál línea quiere traducir (1-19), T(odas), F(in): 1

Bonito es mejor que feo

¿Cuál línea quiere traducir (1-19), T(odas), F(in): 5

Sencillo es mejor que anidado

¿Cuál línea quiere traducir (1-19), T(odas), F(in): t

Bonito es mejor que feo
Explícito es mejor que implícito
Simple es mejor que complejo
Complejo es mejor que complicado
Sencillo es mejor que anidado
Disperso es mejor que denso
La legibilidad cuenta
Los casos especiales no son lo suficientemente especiales para romper las reglas
La practicidad bate a la pureza
Los errores no deben pasar inadvertidos
A menos que sean explícitamente silenciados
En caso de ambigüedad, rechazar la tentación de adivinar
Debe haber una -preferiblemente única- forma obvia de hacerlo.
Aunque es única forma no sea obvia en un primer momento, a no ser que seas holandés
Ahora es mejor que nunca
Aunque nunca es mejor que 'ahora' mismo
Si la implementación es difícil de explicar, es una mala idea
Si la implementación es fácil de explicar, debe ser una buena idea

¿Cuál línea quiere traducir (1-19), T(odas), F(in): f

>>>
```

En el programa se observa el uso de un diccionario mediante la variable *Zen*; para poder leer la línea del diccionario indicada en la variable *n* introducida desde el teclado mediante *input()*, se utiliza una lista generada por comprensión con la función *list()*.