## Tipo diccionarios

El diccionario, define una relación uno a uno entre claves y valores.

|  |  |  |  |
| --- | --- | --- | --- |
| Clase | Tipo | Notas | Ejemplo |
| dict | Mapeos | Mutable, sin orden. | {’cms’:"Plone", ’version’:5} |

Un objeto *mapping* mapea valores *hashable* a objetos arbitrariamente. Los objetos Mapeos son objetos mutables. El diccionario es el único tipo de mapeo estándar actual. Para otro contenedores ver los integrados en las clases “*lista*”, “*conjuntos* (Página 73)”, y “*tupla* ”, y el modulo “collections”.

Los diccionarios pueden ser creados colocando una lista separada por coma de pares “key:value” entre {}, por ejemplo: “{’python’: 27, ’plone’: 51}” o “{27:’python’, 51:’plone’}”, o por el constructor “*dict()*”.

**>>>** diccionario = {

**...** "clave1":234,

**...** "clave2":True,

**...** "clave3":"Valor 1",

**...** "clave4":[1,2,3,4]

**...** }

**>>> print** diccionario, type(diccionario)

{'clave4': [1, 2, 3, 4], 'clave1': 234,

'clave3': 'Valor 1', 'clave2': True} <type 'dict'>

Usted puede acceder a los valores del diccionario usando cada su clave, se presenta unos ejemplos a continuación:

**Tipo diccionarios**

**>>>** diccionario['clave1']

234

**>>>** diccionario['clave2']

True

**>>>** diccionario['clave3']

'Valor 1'

**>>>** diccionario['clave4'] [1, 2, 3, 4]

Un diccionario puede almacenar los diversos tipos de datos integrados en Python usando la función *type(),* usted puede pasar el diccionario con la clave que usted desea determinar el tipo de dato, se presenta unos ejemplos a continuación:

**>>>** type(diccionario['clave1'])

<type 'int'>

**>>>** type(diccionario['clave2'])

<type 'bool'>

**>>>** type(diccionario['clave3'])

<type 'str'>

**>>>** type(diccionario['clave4'])

<type 'list'>

### Operaciones

Los objetos de tipo diccionario permite una serie de operaciones usando operadores integrados en el intérprete Python para su tratamiento, a continuación algunos de estos:

#### Acceder a valor de clave

Esta operación le permite acceder a un valor específico del *diccionario* mediante su clave.

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1, django=2.1)

**>>>** versiones['zope']

2.13

#### Asignar valor a clave

Esta operación le permite asignar el valor específico del *diccionario* mediante su clave.

**>>>** versiones = {'python': 2.7, 'zope': 2.13, 'plone': None}

**>>>** versiones['plone']

**>>>** versiones['plone'] = 5.1

**>>>** versiones

{'python': 2.7, 'zope': 2.13, 'plone': 5.1}

**>>>** versiones['plone']

5.1

#### Iteración in

Este operador es el mismo operador integrado *in* en el intérprete Python pero aplicada al uso de la secuencia de tipo diccionario.

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1, django=2.1)

**>>> print** versiones

{'zope': 2.13, 'python': 2.7, 'plone': 5.1, 'django': 2.1}

**>>>** 'plone' **in** versiones True

**>>>** 'flask' **in** versiones False

En el ejemplo anterior este operador devuelve True si la clave está en el diccionario versiones, de lo contrario devuelve False.

### Métodos

Los objetos de tipo diccionario tienen una serie de métodos integrados a continuación:

#### clear()

Este método remueve todos los elementos desde el diccionario.

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>> print** versiones

{'zope': 2.13, 'python': 2.7, 'plone': 5.1}

**>>>** versiones.clear()

**>>> print** versiones

{}

#### copy()

Este método devuelve una copia superficial del tipo diccionario:

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>>** otro\_versiones = versiones.copy()

**>>>** versiones == otro\_versiones True

#### fromkeys()

Este método crea un nuevo diccionario con *claves* a partir de un tipo de dato *secuencia*. El valor de value por defecto es el tipo *None.*

**>>>** secuencia = ('python', 'zope', 'plone')

**>>>** versiones = dict.fromkeys(secuencia)

**>>> print** "Nuevo Diccionario : *%s*" % str(versiones)

Nuevo Diccionario : {'python': None, 'zope': None, 'plone': None}

En el ejemplo anterior inicializa los valores de cada clave a None, mas puede inicializar un *valor* común por defecto para cada *clave*:

**>>>** versiones = dict.fromkeys(secuencia, 0.1)

**>>> print** "Nuevo Diccionario : *%s*" % str(versiones)

Nuevo Diccionario : {'python': 0.1, 'zope': 0.1, 'plone': 0.1}

#### get()

Este método devuelve el valor en base a una coincidencia de búsqueda en un diccionario mediante una clave, de lo contrario devuelve el objeto *None.*

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>>** versiones.get('plone')

5.1

**>>>** versiones.get('php') >>>

has\_key()

Este método devuelve el valor True si el diccionario tiene presente la clave enviada como argumento.

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>>** versiones.has\_key('plone')

True

**>>>** versiones.has\_key('django') False

#### items()

Este método devuelve una lista de pares de diccionarios (clave, valor), como 2 tuplas.

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>>** versiones.items()

[('zope', 2.13), ('python', 2.7), ('plone', 5.1)]

#### iteritems()

Este método devuelve un *iterador* sobre los elementos (clave, valor) del diccionario. Lanza una excepción *StopIteration* (Página 188) si llega al final de la posición del diccionario.

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>> print** versiones

{'zope': 2.13, 'python': 2.7, 'plone': 5.1}

**>>>** versiones.iteritems()

<dictionary-itemiterator object at 0x7fab9dd4bc58> **>>> for** clave,valor **in** versiones.iteritems():

**... print** clave,valor

**...**

zope 2.13 python 2.7 plone 5.1

**>>>** versionesIterador = versiones.iteritems()

**>>> print** versionesIterador.next()

('zope', 2.13)

**>>> print** versionesIterador.next()

('python', 2.7)

**>>> print** versionesIterador.next()

('plone', 5.1)

**>>> print** versionesIterador.next() Traceback (most recent call last):

File "<stdin>", line 1, in <module> StopIteration

#### iterkeys()

Este método devuelve un *iterador* sobre las claves del diccionario. Lanza una excepción *StopIteration* llega al final de la posición del diccionario.

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>> print** versiones

{'zope': 2.13, 'python': 2.7, 'plone': 5.1}

**>>>** versiones.iterkeys()<dictionary-keyiterator object at 0x7fab9dd4bcb0>

**>>> for** clave **in** versiones.iterkeys():

**... print** clave

**...** zope python plone

**>>>** versionesIterador = versiones.iterkeys()

**>>> print** versionesIterador.next() zope

**>>> print** versionesIterador.next() python

**>>> print** versionesIterador.next() plone

**>>> print** versionesIterador.next() Traceback (most recent call last):

File "<stdin>", line 1, in <module> StopIteration

#### itervalues()

Este método devuelve un *iterador* sobre los valores del diccionario. Lanza una excepción *StopIteration* si llega al final de la posición del diccionario.

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>> print** versiones

{'zope': 2.13, 'python': 2.7, 'plone': 5.1}

**>>>** versiones.itervalues()

<dictionary-valueiterator object at 0x7fab9dd4bc58>

**>>> for** valor **in** versiones.itervalues():

**... print** valor

**...**

2.13

2.7

5.1

**>>>** versionesIterador = versiones.itervalues()

**>>> print** versionesIterador.next()

2.13

**>>> print** versionesIterador.next()

2.7

**>>> print** versionesIterador.next()

5.1

**>>> print** versionesIterador.next() Traceback (most recent call last):

File "<stdin>", line 1, in <module> StopIteration

#### keys()

Este método devuelve una lista de las claves del diccionario:

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>>** versiones.keys()

['zope', 'python', 'plone']

#### pop()

Este método remueve específicamente una clave de diccionario y devuelve valor correspondiente. Lanza una excepción *KeyError* si la clave no es encontrada.

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>>** versiones

{'zope': 2.13, 'python': 2.7, 'plone': 5.1}

**>>>** versiones.pop('zope')

2.13

**>>>** versiones

{'python': 2.7, 'plone': 5.1}

**>>>** versiones.pop('django')

Traceback (most recent call last):

File "<stdin>", line 1, in <module> KeyError: 'django'

#### popitem()

Este método remueve y devuelve algún par (clave, valor) del diccionario como una 2 tuplas. Lanza una excepción *KeyError* si el diccionario esta vacío.

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>>** versiones

{'zope': 2.13, 'python': 2.7, 'plone': 5.1}

**>>>** versiones.popitem()

('zope', 2.13)

**>>>** versiones

{'python': 2.7, 'plone': 5.1}

**>>>** versiones.popitem()

('python', 2.7)

**>>>** versiones

{'plone': 5.1}

**>>>** versiones.popitem()

('plone', 5.1)

**>>>** versiones

{}

**>>>** versiones.popitem()

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

KeyError: 'popitem(): dictionary is empty'

#### setdefault()

Este método es similar a *get(key, default\_value)* pero además asigna la clave key al valor por default\_value para la clave si esta no se encuentra en el diccionario.

D.setdefault(key[,default\_value])

A continuación un ejemplo de cómo trabaja el método setdefault() cuando la clave está en el diccionario:

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>>** zope = versiones.setdefault('zope')

**>>> print** 'Versiones instaladas:', versiones

Versiones instaladas: {'zope': 2.13, 'python': 2.7, 'plone': 5.1}

**>>> print** 'Versión de Zope:', zope

Versión de Zope: 2.13

A continuación un ejemplo de como trabaja el método setdefault() la clave no esta en el diccionario:

**>>>** paquetes = {'python': 2.7, 'zope': 2.13}

**>>> print** paquetes

{'python': 2.7, 'zope': 2.13}

**>>>** plone = paquetes.setdefault('plone')

**>>> print** 'paquetes: ', paquetes paquetes: {'python': 2.7, 'zope': 2.13, 'plone': None}

**>>> print** 'plone: ', plone plone: None

Si el valor no es proveído, el valor default\_value será el tipo objeto integrado *None*

A continuación un ejemplo de cómo trabaja el método setdefault() la clave no está en el diccionario pero esta vez el default\_value es proveído:

**>>>** pkgs = {'python': 2.7, 'zope': 2.13, 'plone': None}

**>>> print** pkgs

{'python': 2.7, 'zope': 2.13, 'plone': None}

**>>>** django = paquetes.setdefault('django', 2.1)

**>>> print** 'paquetes =', pkgs paquetes = {'python': 2.7, 'zope': 2.13, 'plone': None}

**>>> print** 'django =', django django = 2.1

A continuación otro ejemplo en donde puedes agrupar N *tuplas* (Página 61) por el valor el cual se repite más y construir un diccionario que cuyas claves son los valores más repetidos y cuyos valores estén agrupados en tipo *listas:*

**>>>** PKGS = (('zope', 'Zope2'),

**...** ('zope', 'pytz'),

|  |  |
| --- | --- |
| **...** | ('plone', 'Plone'), |
| **...** | ('plone', 'diazo'), |
| **...** | ('plone', 'z3c.form'),) |

>>>

**>>>** paquetes = {}

**>>> for** clave, valor **in** PKGS:

**... if** paquetes.has\_key(clave): **...** paquetes[clave].append(valor) **... else**:

**...** paquetes[clave] = [valor]

**...**

**>>> print** paquetes

{'zope': ['Zope2', 'pytz'], 'plone': ['Plone', 'diazo', 'z3c.form']}

En el tipo tupla PKGS los elementos más repetidos son ’zope’ y ’plone’ estos se convierten en clave del diccionario paquetes y los otros elementos se agrupan en listas como sus respectivos valores.

A continuación un mejor aprovechamiento implementando el método setdefault():

**>>>** PKGS = (('zope', 'Zope2'),

**...** ('zope', 'pytz'),

|  |  |
| --- | --- |
| **...** | ('plone', 'Plone'), |
| **...** | ('plone', 'diazo'), |
| **...** | ('plone', 'z3c.form'),) |

**>>>** paquetes = {}

**>>> for** clave, valor **in** PKGS:

**...** paquetes.setdefault(clave, []).append(valor) **...**

**>>> print** paquetes

{'zope': ['Zope2', 'pytz'], 'plone': ['Plone', 'diazo', 'z3c.form']}

En el ejemplo anterior puede ver que el aprovechamiento del método setdefault() a comparación de no usar el respectivo método.

#### update()

Este método actualiza un diccionario agregando los pares clave-valores en un segundo diccionario. Este método no devuelve nada.

El método update() toma un diccionario o un objeto iterable de pares clave/valor (generalmente tuplas). Si se llama a update() sin pasar parámetros, el diccionario permanece sin cambios.

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>> print** versiones

{'zope': 2.13, 'python': 2.7, 'plone': 5.1}

**>>>** versiones\_adicional = dict(django=2.1)

**>>> print** versiones\_adicional

{'django': 2.1}

**>>>** versiones.update(versiones\_adicional)

Como puede apreciar este método no devuelve nada, más si muestra de nuevo el diccionario versiones puede ver que este fue actualizado con el otro diccionario versiones\_adicional.

**>>> print** versiones

{'zope': 2.13, 'python': 2.7, 'plone': 5.1, 'django': 2.1}

#### values()

Este método devuelve una lista de los valores del diccionario:

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>>** versiones.values()

[2.13, 2.7, 5.1]

#### viewitems()

Este método devuelve un objeto como un conjunto mutable proveyendo una vista en los elementos del diccionario:

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>>** versiones.viewkeys() dict\_keys(['zope', 'python', 'plone']) **>>> for** clave,valor **in** versiones.iteritems():

**... print** clave,valor

**...**

zope 2.13 python 2.7 plone 5.1

#### viewkeys()

Este método devuelve un objeto proveyendo una vista de las claves del diccionario.

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>>** versiones.viewkeys() dict\_keys(['zope', 'python', 'plone']) **>>> for** clave **in** versiones.viewkeys():

**... print** clave

**...** zope python plone

#### viewvalues()

Este método devuelve un objeto proveyendo una vista de los valores del diccionario.

versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>>** versiones.viewvalues() dict\_values([2.13, 2.7, 5.1])

**>>> for** valor **in** versiones.viewvalues():

**... print** valor

**...**

2.13

2.7

5.1

### Funciones

Los objetos de tipo diccionario tienen disponibles una serie de *funciones* integradas en el intérprete Python para su tratamiento, a continuación algunas de estas:

#### cmp()

Esta función es la misma función integrada *cmp()* en el intérprete Python pero aplicada al uso de la secuencia de tipo diccionario.

**>>>** versiones\_proyecto1 = dict(python=2.7, zope=2.13, plone=5.1)

**>>>** versiones\_proyecto2 = dict(django=2.1, django-filter=1.1.0)

**>>> print** cmp(versiones\_proyecto1, versiones\_proyecto2)

1

La función cmp() es usado en Python para comparar valores y claves de dos diccionarios. Si la función devuelve el valor 0 si ambos diccionarios son igual, devuelve el valor 1 si el primer diccionario es mayor que el segundo diccionario y devuelve el valor -1 si el primer diccionario es menor que el segundo diccionario.

#### len()

Esta función es la misma función integrada *len()*en el intérprete Python pero aplicada al uso de la secuencia de tipo diccionario.

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1)

**>>>** len(versiones)

3

### Sentencias

Los objetos de tipo diccionario tienen disponibles una serie de *sentencias* integradas en el intérprete Python para su tratamiento, a continuación algunas de estas:

#### del

Esta sentencia es la misma sentencia integrada *del* en el intérprete Python pero aplicada al uso de la secuencia de tipo diccionario.

**>>>** versiones = dict(python=2.7, zope=2.13, plone=5.1, django=2.1)

**>>> print** versiones

{'zope': 2.13, 'python': 2.7, 'plone': 5.1, 'django': 2.1}

**>>> del** versiones['django']

**>>> print** versiones

{'zope': 2.13, 'python': 2.7, 'plone': 5.1}

En el código fuente anterior se usa la sentencia del para eliminar un elemento del diccionario mediante su respectiva clave.

### Convertir a diccionarios

Para convertir a *tipos diccionarios* debe usar la función *dict()*la cual *está integrada* en el intérprete Python.

### Ejemplos

A continuación, se presentan un ejemplo de su uso: Ejemplo de definición de un diccionario

datos\_basicos = {

"nombres":"Leonardo Jose",

"apellidos":"Caballero Garcia",

"cedula":"26938401",

"fecha\_nacimiento":"03/12/1980",

"lugar\_nacimiento":"Maracaibo, Zulia, Venezuela",

"nacionalidad":"Venezolana",

"estado\_civil":"Soltero" }

Ejemplo de operaciones con tipo diccionario con funciones propias

**print** "**\n**Claves de diccionario:", datos\_basicos.keys() **print** "**\n**Valores de diccionario:", datos\_basicos.values() **print** "**\n**Elementos de diccionario:", datos\_basicos.items()

Ejemplo de iteración avanzada sobre diccionarios con función iteritems

**for** key, value **in** datos\_basicos.iteritems(): **print**('Clave: *%s*, tiene el valor: *%s*' % (key, value))

Ejemplo real de usar tipo diccionario

**print** "**\n\n**Inscripción de Curso" **print** "===================="

**print** "**\n**Datos de participante" **print** "---------------------"

**print** "Cédula de identidad: ", datos\_basicos['cedula'] **print** "Nombre completo: " + datos\_basicos['nombres'] + " " + \ datos\_basicos['apellidos'] **import datetime**, **locale**, **os**

locale.setlocale(locale.LC\_ALL, os.environ['LANG']) **print** "Fecha y lugar de nacimiento:", datetime.datetime.strftime( datetime.datetime.strptime( datos\_basicos['fecha\_nacimiento'], '*%d*/%m/%Y'), "*%d* de %B de %Y" ) + " en " + datos\_basicos['lugar\_nacimiento'] + "." **print** "Nacionalidad:", datos\_basicos['nacionalidad'] **print** "Estado civil:", datos\_basicos['estado\_civil']

### Ayuda integrada

Usted puede consultar toda la documentación disponible sobre los diccionarios desde la *consola interactiva* de la siguiente forma:

help(dict)