Dictionaries

WE'LL COVER THE FOLLOWING

A dictionary is an unordered set of key-value pairs. When you add a key to a dictionary, you must also add a value for that key. (You can always change the value later.) Python dictionaries are optimized for retrieving the value when you know the key, but not the other way around.

> *A dictionary in Python is like a hash in Perl 5. In Perl 5, variables that store hashes always start with a % character. In Python, variables can be named anything, and Python keeps track of the datatype internally.*

Creating A Dictionary

Creating a dictionary is easy. The syntax is similar to sets, but instead of values, you have key-value pairs. Once you have a dictionary, you can look up values by their key.

```
a_dict = {'server': 'db.diveintopython3.org', 'database': 'mysql'}          #①
print (a_dict)
#{'server': 'db.diveintopython3.org', 'database': 'mysql'}

print (a_dict['server'] )                                                   #②
#db.diveintopython3.org

print (a_dict['database'] )                                                 #③
#mysql
```

```
print (a_dict['db.diveintopython3.org'])                    #④
#Traceback (most recent call last):
# File "/usercode/__ed_file.py", line 11, in <module>
# print (a_dict['db.diveintopython3.org']) #\u2463
#KeyError: 'db.diveintopython3.org'
```

① First, you create a new dictionary with two items and assign it to the variable `a_dict`. Each item is a key-value pair, and the whole set of items is enclosed in curly braces.

② `'server'` is a key, and its associated value, referenced by `a_dict['server']`, is `'db.diveintopython3.org'`.

③ `'database'` is a key, and its associated value, referenced by `a_dict['database']`, is `'mysql'`.

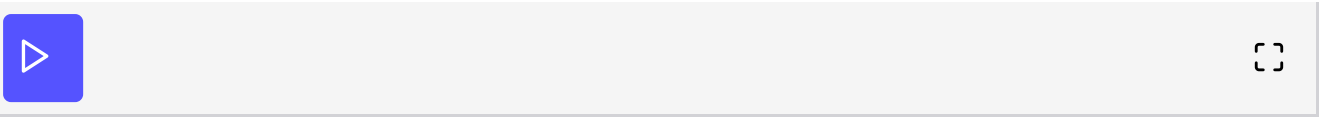
④ You can get values by key, but you can't get keys by value. So `a_dict['server']` is `'db.diveintopython3.org'`, but `a_dict['db.diveintopython3.org']` raises an exception, because `'db.diveintopython3.org'` is not a key.

Modifying A Dictionary

Dictionaries do not have any predefined size limit. You can add new key-value pairs to a dictionary at any time, or you can modify the value of an existing key. Continuing from the previous example:

```
a_dict = {'server': 'db.diveintopython3.org', 'database': 'mysql'}
print (a_dict)
#{'server': 'db.diveintopython3.org', 'database': 'mysql'}

a_dict['database'] = 'blog'  #①
print (a_dict)
#{'server': 'db.diveintopython3.org', 'database': 'blog'}

a_dict['user'] = 'mark'      #②
print (a_dict )              #③
#{'server': 'db.diveintopython3.org', 'database': 'blog', 'user': 'mark'}

a_dict['user'] = 'dora'      #④
print (a_dict)
#{'server': 'db.diveintopython3.org', 'database': 'blog', 'user': 'dora'}

a_dict['User'] = 'mark'      #⑤
print (a_dict)
#{'server': 'db.diveintopython3.org', 'User': 'mark', 'database': 'blog', 'user': 'dora'}
```

```
#{ server': 'db.diveintopython3.org', 'user': 'mark', 'database': 'blog', 'user': 'dora' }
```



① You can not have duplicate keys in a dictionary. Assigning a value to an existing key will wipe out the old value.

② You can add new key-value pairs at any time. This syntax is identical to modifying existing values.

③ The new dictionary item (key `'user'`, value `'mark'`) appears to be in the middle. In fact, it was just a coincidence that the items appeared to be in order in the first example; it is just as much a coincidence that they appear to be out of order now.

④ Assigning a value to an existing dictionary key simply replaces the old value with the new one.

⑤ Will this change the value of the user key back to “mark”? No! Look at the key closely — that’s a capital `U` in `"User"`. Dictionary keys are case-sensitive, so this statement is creating a new key-value pair, not overwriting an existing one. It may look similar to you, but as far as Python is concerned, it’s completely different.

Mixed-Value Dictionaries

Dictionaries aren’t just for strings. Dictionary values can be any datatype, including integers, booleans, arbitrary objects, or even other dictionaries. And within a single dictionary, the values don’t all need to be the same type; you can mix and match as needed. Dictionary keys are more restricted, but they can be strings, integers, and a few other types. You can also mix and match key datatypes within a dictionary.

In fact, you’ve already seen a dictionary with non-string keys and values, in your first Python program.

```
SUFFIXES = {1000: ['KB', 'MB', 'GB', 'TB', 'PB', 'EB', 'ZB', 'YB'],
            1024: ['KiB', 'MiB', 'GiB', 'TiB', 'PiB', 'EiB', 'ZiB', 'YiB']}
```

Let’s tear that apart in the interactive shell.

```
SUFFIXES = {1000: ['KB', 'MB', 'GB', 'TB', 'PB', 'EB', 'ZB', 'YB'],
            1024: ['KiB', 'MiB', 'GiB', 'TiB', 'PiB', 'EiB', 'ZiB', 'YiB']}

print (len(SUFFIXES))      #①
#2

print (1000 in SUFFIXES)   #②
#True

print (SUFFIXES[1000] )    #③
#['KB', 'MB', 'GB', 'TB', 'PB', 'EB', 'ZB', 'YB']

print (SUFFIXES[1024])     #④
#['KiB', 'MiB', 'GiB', 'TiB', 'PiB', 'EiB', 'ZiB', 'YiB']

print (SUFFIXES[1000][3])  #⑤
#'TB'
```

① Like lists and sets, the `len()` function gives you the number of keys in a dictionary.

② And like lists and sets, you can use the in operator to test whether a specific key is defined in a dictionary.

③ `1000` is a key in the `SUFFIXES` dictionary; its value is a list of eight items (eight strings, to be precise).

④ Similarly, `1024` is a key in the `SUFFIXES` dictionary; its value is also a list of eight items.

⑤ Since `SUFFIXES[1000]` is a list, you can address individual items in the list by their 0-based index.

Dictionaries In A Boolean Context

Empty dictionaries are false; all other dictionaries are true.

You can also use a dictionary in a boolean context, such as an if statement.

```
def is_it_true(anything):
   if anything:
     print("yes, it's true")
   else:
     print("no, it's false")
```

```
print (is_it_true({}) )            #①
#no, it's false
#None
print (is_it_true({'a': 1}))       #②
#yes, it's true
#None
```

① In a boolean context, an empty dictionary is false.

② Any dictionary with at least one key-value pair is true.