10 - Python Dictionary

John R. Woodward

Dictionary 1

1. A dictionary is **mutable** and is another container type that can store any number of Python objects, including other container types.
2. Dictionaries consist of **pairs** (called items) of **keys** and their corresponding **values**.
3. Think of the **key to a value**
4. Python dictionaries are also known as associative arrays or hash tables.
5. dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}

Keys

- Each **key** is separated from its value by a colon (**:**), the items are separated by commas, and the whole thing is enclosed in **curly braces**.
- An **empty dictionary** without any items is written with just two curly braces, like this: **{}**.
- **Keys** are **unique** within a dictionary while values may not be. The **values** of a dictionary can be of **any type**, but the **keys** must be of an **immutable** data type such as **strings, numbers, or tuples**.

Example

```
digitsStrings = {1: "one",
 2: "two", 3:"three"}
print digitsStrings
digitsStrings[4] = "four"
print digitsStrings
del digitsStrings[1]
print digitsStrings
digitsStrings["five"] = 5
print digitsStrings
digitsStrings[1.2]="one.two"
print digitsStrings
```

{1: 'one', 2: 'two', 3: 'three'}

{1: 'one', 2: 'two',
3: 'three', 4: 'four'}

{2: 'two', 3: 'three', 4: 'four'}

{'five': 5, 2: 'two',
3: 'three', 4: 'four'}

{'five': 5, 2: 'two', 3: 'three',
4: 'four', 1.2: 'one.two'}

Keys – must be unique

- Keys must be **unique**
- One key accesses **one value**.
- I would keep **the same datatype**.(string,int)
- I would **not use floats.**
- Which are possible keys
  - Student ID number,
  - date of birth,
  - email address,
  - car registration,
  - full name,

Accessing Values in Dictionary

1. To access dictionary elements, you can use the familiar square brackets.
2. dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
   print "dict['Name']: ", dict['Name'];
3. print "dict['Age']: ", dict['Age'];
4. it produces **the following result**:
5. dict['Name']: Zara
6. dict['Age']: 7

Updating Dictionary

You can **update** a dictionary by **adding** a new entry or item (i.e., a key-value pair), **modifying** an existing entry, or **deleting** an existing entry as shown below in the simple example:

- dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
- dict['Age'] = 8; # update existing entry
- dict['School'] = "DPS School"; # Add new entry
- print "dict['Age']: ", dict['Age'];
- print "dict['School']: ", dict['School'];

it produces **the following result**:

- dict['Age']: 8
- dict['School']: DPS School

Delete Dictionary Elements:

- dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'};
- del dict['Name']; # remove entry with key 'Name'
- dict.clear(); # remove all entries in dict del
- dict ; # delete entire dictionary

Clear or delete a dictionary

- `dictNumbers = {1:"one", 2:"two", 3:"three"}`
- `print dictNumbers`
- `dictNumbers.clear()`
- `print dictNumbers`
- `del dictNumbers`
- `print dictNumbers`

output

- {1: 'one', 2: 'two', 3: 'three'}
- {}
- `print dictNumbers`
- `NameError: name 'dictNumbers' is not defined`

Properties of Dictionary Values:

- Dictionary **values** have no restrictions.
- They can be any **arbitrary Python object**, either standard objects or user-defined objects.
- However, same is not true for the keys.

Properties of Dictionary Keys:

- There are two important points to remember about dictionary keys:
- **(a)** More than one entry per key not allowed. Which means **no duplicate key is allowed**. When duplicate keys encountered during assignment, **the last assignment wins**. Following is a simple example:

Example – do not do

```
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'};
print "dict['Name']: ", dict['Name'];
```

it produces the following result:

- dict['Name']: Manni
- Note that we lost “'Name': 'Zara',”

Keys must be immutable

- **(b)** you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.
- dict = {['Name']: 'Zara', 'Age': 7};
- print "dict['Name']: ", dict['Name'];
- it produces **the following result**:
- Traceback (most recent call last): File "test.py", line 3, in <module> dict = {['Name']: 'Zara', 'Age': 7}; **TypeError: list objects are unhashable**

Built-in Dictionary Functions

cmp(dict1, dict2) Compares elements of both dict.
len(dict) Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
str(dict) Produces a printable string representation of a dictionary
type(variable) Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

Built-in Dictionary Methods

dict.clear() Removes all elements of dictionary *dict*
dict.copy() Returns a shallow copy of dictionary *dict*
dict.fromkeys() Create a new dictionary with keys from seq and values *set* to *value*.
dict.get(key, default=None) For *key* key, returns value or default if key not in dictionary
dict.has_key(key) Returns *true* if key in dictionary *dict*, *false* otherwise
dict.items() Returns a list of *dict*'s (key, value) tuple pairs
dict.keys() Returns list of dictionary dict's keys
dict.setdefault(key, default=None) Similar to get(), but will set dict[key]=default if *key* is not already in dict
dict.update(dict2) Adds dictionary *dict2*'s key-values pairs to *dict*
dict.values() Returns list of dictionary *dict*'s values

What does the following print?

- dict1 = {"key1": "value1", "key2": "value2"}
- dict2 = dict1
- print dict2
- dict2["key2"] = "WHY?!"
- print dict1

What does the following print?

- dict1 = {"key1": "value1", "key2": "value2"}
- dict2 = dict1
- print dict2
- dict2["key2"] = "WHY?!"
- print dict1

output

{'key2': 'value2', 'key1': 'value1'}
dict2["key2"] = "WHY?!"
print dict1
{'key2': 'WHY?!', 'key1': 'value1'}

Copy a dictionary

- If you want to copy the dict (**which is rare**), you have to do so explicitly with

- dict2 = dict(dict1)
- or
- dict2 = dict1.copy()

What is the output?

```
def inc(x):
    print "inc" ,x
    x = x +1
    print "inc" ,x
y = 99
print "outside function ", y
inc(y)
print "outside function ", y
```

What is the output?

```
def inc(x):
    print "inc" ,x
    x = x +1
    print "inc" ,x
y = 99
print "outside function ", y
inc(y)
print "outside function ", y
```

output

outside function 99
inc 99
inc 100
outside function 99

What is the output?

```
def incList(x):
    print "inc" ,x
    x[0] = x[0] + 1
    print "inc" ,x
y = [99]
print "outside function ", y
incList(y)
print "outside function ", y
```

What is the output?

```
def incList(x):
    print "inc" ,x
    x[0] = x[0] + 1
    print "inc" ,x
y = [99]
print "outside function ", y
incList(y)
print "outside function ", y
```

output

outside function [99]
inc [99]
inc [100]
outside function [100]

What is the output?

```
def inc(x):
    print "inc" ,x
    x = x +1
    print "inc" ,x
    return x #added
y = 99
print "outside function ", y
y = inc(y) #assignment
print "outside function ", y
```

What is the output?

```
def inc(x):
    print "inc" ,x
    x = x +1
    print "inc" ,x
y = 99
print "outside function ", y
inc(y)
print "outside function ", y
```

output

outside function 99
inc 99
inc 100
outside function 99

Tuples

John R. Woodward

Tuples 1

1. A tuple is a sequence of **immutable** Python objects.
2. Tuples are **sequences**, just like lists.
3. The only difference is that tuples **can't be changed** i.e., tuples are immutable
4. tuples use **parentheses** and lists use square brackets.
5. Creating a tuple is as simple as putting different **comma-separated values** and optionally you can put these comma-separated values between parentheses also.
6. tup1 = ('physics', 'chemistry', 1997, 2000);
7. tup2 = (1, 2, 3, 4, 5);
8. tup3 = "a", "b", "c", "d";

Tuples 2

1. The **empty tuple** is written as two parentheses containing nothing:
2. tup1 = **()**
3. To write a tuple containing a single value **you have to include a comma**, even though there is only one value:
4. tup1 = (50,);
5. ??? tup1 = (50);??? What would this mean
6. Like string indices, tuple indices start at 0, and tuples can be sliced, concatenated and so on.

Types - Be careful

- `tup1 = (50,)`
- `print tup1`
- `print type(tup1)`
- `tup1 = (50)#no comma`
- `print tup1`
- `print type(tup1)`

Output:

(50,)

<type 'tuple'>

50

<type 'int'>

Same with Strings

```
tup1 = ("Stirling")
print tup1
print type(tup1)
tup1 = ("Stirling",)
print tup1
print type(tup1)
```

OUTPUT

Stirling
<type 'str'>
('Stirling',)
<type 'tuple'>

Updating Tuples

1. Tuples are **immutable**.
2. You are able to take portions of existing tuples to create new tuples
3. tup1 = (12, 34.56);
4. tup2 = ('abc', 'xyz');
5. tup3 = tup1 + tup2;
6. print tup3;
7. it produces the following result:
8. (12, 34.56, 'abc', 'xyz')

Delete Tuple Elements

1. Removing individual tuple elements **is not possible**.
2. There is nothing wrong with putting together another tuple with the desired elements.
3. To explicitly remove an entire tuple, just use the **del** statement.

```
4. tup = ('physics', 'chemistry', 1997, 2000);
5. print tup;
6. del tup;
7. print "After deleting tup : "
8. print tup;#WHAT ERROR WOULD YOU GET
```

Basic Tuples Operations

Python Expression	Results	Description
len((1, 2, 3))	3	**Length**
(1, 2, 3) **+** (4, 5, 6)	(1, 2, 3, 4, 5, 6)	**Concatenation**
('Hi!',) ***** 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	**Repetition**
3 **in** (1, 2, 3) 4 not in (1, 2, 3)	True True	**Membership**
for x **in** (1, 2, 3): print x,	1 2 3	**Iteration**

Indexing

- L = ('spam', 'Spam', 'SPAM!')

Python Expression	Results	Description
L[2]	'SPAM!'	Offsets start at zero
L[-2]	'Spam'	Negative: count from the right
L[1:]	['Spam', 'SPAM!']	Slicing fetches sections

Built-in Tuple Functions

cmp(tuple1, tuple2) Compares elements of both tuples.
len(tuple) Gives the total length of the tuple.
max(tuple) Returns item from the tuple with max value.
min(tuple) Returns item from the tuple with min value.
tuple(seq) Converts a list into tuple.

Accessing Values in Tuples

- To access values in tuple, use the **square brackets**.
- tup1 = ('physics', 'chemistry', 1997, 2000);
- tup2 = (1, 2, 3, 4, 5, 6, 7);
- print "tup1[0]: ", tup1[0]
- print "tup2[1:5]: ", tup2[1:5]
- it produces **the following result**:
- tup1[0]: physics
- tup2[1:5]: [2, 3, 4, 5]

EXAMPLE

- `integers = (0,1,2,3,4,5,6,7,8,9)`
- `print integers`
- `print integers[1]`
- `print integers[-1]`
- `subset1 = integers[1:3]`
- `print subset1`

EXAMPLE

- `integers = (0,1,2,3,4,5,6,7,8,9)`
- `print integers`
- `print integers[1]`
- `print integers[-1]`
- `subset1 = integers[1:3]`
- `print subset1`

(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
1
9
(1, 2)

OUTPUT

EXAMPLE

```
subset1 = integers[1:8:2]
print subset1
subset1 = integers[8:1:2]
print subset1
subset1 = integers[8:4:-1]
print subset1
subset1 = integers[4:8:-1]
print subset1
```

EXAMPLE

```
subset1 = integers[1:8:2]
print subset1
subset1 = integers[8:1:2]
print subset1
subset1 = integers[8:4:-1]
print subset1
subset1 = integers[4:8:-1]
print subset1
```

output

(1, 3, 5, 7)

()

(8, 7, 6, 5)

()