

Fall, 2020

## Python Programming Workshop

# Workshop 3

# Tuples and Dictionaries

Kevin Kam Fung Yuen  
Department of Computing  
Hong Kong Polytechnic University  
[kevin.yuen@polyu.edu.hk](mailto:kevin.yuen@polyu.edu.hk)

## Class Schedule

19 Sep: Python Programming Environment and Basics

26 Sep: Control Flow and Lists

3 Oct: **Lists, Tuples and Dictionaries**

10 Oct: Functions, Modules, OO and Packages

17 Oct: Scientific Computing and Plotting

Lecture notes are available in

<https://github.com/kkfyuen/PythonWorkshop2020>

# References

- Toby Donaldson, Starting out with Python, 2021 (2014)
- Zed A. Shaw, Learn Python 3 **the Hard Way**: A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code, July 2017
  - <https://learnpythonthehardway.org/python3/>
  - <http://www.informit.com/promotions/book-registration-learn-python-3-the-hard-way-141409>  
(videos)

## Outline

- Some useful tips for swap and list
- Tuple: index, subset, unpack, iterate elements
- Immutable nature of tuple
- Dictionary: index, get, add, remove, sort
- For loop and Dictionary

# Some useful tips in Python

## Swap the values

```
>>> # swap the values of x and y
>>>
>>> x = 30
>>>
>>> y = 50
>>> # traditional method
>>> # add a temporary variable h to handle
>>>
>>> h = x
>>> x = y
>>> y = h
>>> print(x, y)
50 30
```

```
>>> # swap the values of x and y
>>>
>>> x = 30
>>>
>>> y = 50
>>> #python method
>>>
>>> x,y = y,x
>>> print(x,y)
50 30
_
```

## Create a List with for loop and condition

```
>>> [a for a in range(10)]  
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> [a for a in range(1,10)]  
[1, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> [a for a in range(1,10,2)]  
[1, 3, 5, 7, 9]  
>>> [a for a in range(0,10,2)]  
[0, 2, 4, 6, 8]  
>>> [a for a in range(3,13) if a%2 == 0]  
[4, 6, 8, 10, 12]
```



# List with map, filter and reduce

```
>>> import math
>>> x = [2, 4, 7, 9, 20]
>>>
>>> math.sqrt(2)
1.4142135623730951
>>> math(x) # error
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'module' object is not callable
```

reduce function

```
>>> from functools import reduce
>>> x = [2, 4, 7, 9, 20]
>>> def divide(a,b):
...     return a/b
...
>>> reduce(divide,x)
0.0003968253968253968
>>> # compare the results
>>> 2/4/7/9/20
0.0003968253968253968
```

map function

```
>>> map(math.sqrt, x)
<map object at 0x000001C06CC96640>
>>> list(map(math.sqrt, x))
[1.4142135623730951, 2.0, 2.6457513110645907, 3.0, 4.47213595499958]
```

filter function

```
>>> x = [2, 4, 7, 9, 20]
>>> def odd(a):
...     return a % 2 != 0
...
>>> list(map(odd, x))
[False, False, True, True, False]
>>> list(filter(odd, x))
[7, 9]
```



# Lambda

# A lambda function can take any number of arguments, but can only have one expression.

lambda arguments : expression

# when we use a lambda function inside an other function, we can use the same function definition to make both functions

```
>>> x = lambda a : a - 2.5
>>> print(x(1))
-1.5
>>> x = lambda a, b : a ** b
>>> print(x(2, 5))
32
```

```
>>> def myfunc(m):
...     return lambda x : x ** m
...
>>> power2 = myfunc(2)
>>> power3 = myfunc(3)
>>> power2
<function myfunc.<locals>.<lambda> at 0x000001D39E29D5E0>
>>> print(power2(3))
9
>>> print(power3(3))
27
```

# List with Lambda, map, filter and reduce

```
>>> x = [2, 4, 7, 9, 20]
>>> def odd(a):
...     return a % 2 != 0
...
>>> list(map(odd, x))
[False, False, True, True, False]
>>> list(filter(odd, x))
[7, 9]
```

```
>>> from functools import reduce
>>> x = [2, 4, 7, 9, 20]
>>> def divide(a,b):
...     return a/b
...
>>> reduce(divide,x)
0.0003968253968253968
>>> # compare the results
>>> 2/4/7/9/20
0.0003968253968253968
```

```
>>> map(math.sqrt, x)
<map object at 0x000001C06CC96640>
>>> list(map(math.sqrt, x))
[1.4142135623730951, 2.0, 2.6457513110645907, 3.0, 4.47213595499958]
```

```
>>> from functools import reduce
>>> x = [2, 4, 7, 9, 20]
>>>
>>> list(map(lambda a : a**(1/2),x))
[1.4142135623730951, 2.0, 2.6457513110645907, 3.0, 4.47213595499958]
>>> list(filter(lambda a : a % 2 != 0, x))
[7, 9]
>>> reduce(lambda a,b : a/b, x)
0.0003968253968253968
```

## Three data types

- Square brackets [ ] : list
- Round brackets ( ) or no brackets: tuple
- Curly brackets { } : dictionary

# 1. Operations on Tuples

# 2. Immutable Nature of Tuples

## Tuples may be constructed in a number of ways:

- Using a pair of parentheses to denote the empty tuple: ()
- Using a trailing comma for a singleton tuple: a, or (a,)
- Separating items with commas: a, b, c or (a, b, c)
- Using the [tuple\(\)](#) built-in: tuple() or tuple(iterable)

```
>>> A = (1, 2, 3, 4)
>>> A
(1, 2, 3, 4)
>>> B = (1, )
>>> B
(1,)
>>> C = (10)
>>> C
10
>>> A * 2
(1, 2, 3, 4, 1, 2, 3, 4)
>>>
```

```
>>> tuple("abcd")
('a', 'b', 'c', 'd')
>>> tuple([1, 2, 3, 4])
(1, 2, 3, 4)
>>>
```

```
>>> A + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate tuple (not "int") to tuple
>>>
```



## Discussion 1 (L3D1)

Among the Python data types we have learned so far, which are immutable and which are mutable?

**Note:**

- Because tuples are immutable, they are more efficient in terms of performance and memory use.
- Tuples are useful in situations where you want to share the data with others but not allow them to modify the data. They can use the data values, but no change is reflected in the original data shared.

```
>>> a = 4
>>> a
4
>>> a = a + 1
>>> a
5
>>> a = a + 2
>>> a
7
>>>
```

```
>>> B[1]
2
>>> B = [1, 2, 3, 4]
>>> B[1]
2
>>> B[1] = 20
>>> B[1]
20
>>>
```

```
>>> A = (1, 2, 3, 4)
>>> A[1]
2
>>> A[1] = 20
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> A[1]
2
>>>
```

## Quiz 1 (L3Q1)

The variable `grades` refers to `(70,80,90)`. What does each of the following evaluate to?

`grades[-1]`

`grades[grades.index(90)]`

`grades.pop()`

`grades[len(grades)-1]`

```
>>> grades = (70, 80, 90)
>>> grades[-1]
90
>>> grades.index(90)
2
>>> grades[grades.index(90)]
90
>>> grades.pop()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'pop'
>>> len(grades)
3
>>> grades[len(grades)-1]
90
>>>
```



# 3. Operations on Dictionaries

# Dictionaries

- Dictionary: object that stores a collection of data
  - Each element consists of a key and a value
    - Often referred to as *mapping* of key to value
    - **Key** must be an *immutable object*
  - To retrieve a specific value, use the key associated with it
  - Format for creating a dictionary

```
dictionary =  
    {key1:val1, key2:val2}
```

## Retrieving a Value from a Dictionary

- Elements in dictionary are unsorted
- General format for retrieving value from dictionary:  
`dictionary[key]`
  - If `key` in the dictionary, associated value is returned, otherwise, `KeyError` exception is raised
- Test whether a key is in a dictionary using the `in` and `not in` operators
  - Helps prevent `KeyError` exceptions

## Adding Elements to an Existing Dictionary

- Dictionaries are mutable objects
- To add a new key-value pair:

*dictionary[key] = value*

- If key exists in the dictionary, the value associated with it will be changed

# Deleting Elements From an Existing Dictionary

- To delete a key-value pair:

```
del dictionary[key]
```

- If key is not in the dictionary, `KeyError` exception is raised

# Getting the Number of Elements and Mixing Data Types

- len function: used to obtain number of elements in a dictionary
- Keys must be immutable objects, but associated values can be any type of object
  - One dictionary can include keys of several different immutable types
- Values stored in a single dictionary can be of different types

# Creating an Empty Dictionary and Using `for` Loop to Iterate Over a Dictionary

- To create an empty dictionary:
  - Use `{ }`
  - Use built-in function `dict()`
  - Elements can be added to the dictionary as program executes
- Use a `for` loop to iterate over a dictionary
  - General format: `for key in dictionary:`



## Some Dictionary Methods

- clear method: deletes all the elements in a dictionary, leaving it empty
  - Format: `dictionary.clear()`
- get method: gets a value associated with specified key from the dictionary
  - Format: `dictionary.get(key, default)`
    - `default` is returned if `key` is not found
  - Alternative to `[]` operator
    - Cannot raise `KeyError` exception

## Some Dictionary Methods (cont'd.)

- items method: returns all the dictionaries keys and associated values
  - Format: `dictionary.items()`
  - Returned as a *dictionary view*
    - Each element in dictionary view is a tuple which contains a key and its associated value
    - Use a `for` loop to iterate over the tuples in the sequence
      - Can use a variable which receives a tuple, or can use two variables which receive key and value

## Some Dictionary Methods (cont'd.)

- keys method: returns all the dictionaries keys as a sequence
  - Format: `dictionary.keys()`
- pop method: returns value associated with specified key and **removes** that key-value pair from the dictionary
  - Format: `dictionary.pop(key, default)`
    - `default` is returned if `key` is not found

## Some Dictionary Methods (cont'd.)

- popitem method: returns a randomly selected key-value pair and removes that key-value pair from the dictionary
  - Format: `dictionary.popitem()`
  - Key-value pair returned as a tuple
- values method: returns all the dictionaries values as a sequence
  - Format: `dictionary.values()`
  - Use a `for` loop to iterate over the values

## Some Dictionary Methods (cont'd.)

**Table 9-1** Some of the dictionary methods

Method	Description
<code>clear</code>	Clears the contents of a dictionary.
<code>get</code>	Gets the value associated with a specified key. If the key is not found, the method does not raise an exception. Instead, it returns a default value.
<code>items</code>	Returns all the keys in a dictionary and their associated values as a sequence of tuples.
<code>keys</code>	Returns all the keys in a dictionary as a sequence of tuples.
<code>pop</code>	Returns the value associated with a specified key and removes that key-value pair from the dictionary. If the key is not found, the method returns a default value.
<code>popitem</code>	Returns a randomly selected key-value pair as a tuple from the dictionary and removes that key-value pair from the dictionary.
<code>values</code>	Returns all the values in the dictionary as a sequence of tuples.

## ex39

```
>>> #List
... things = ['a', 'b', 'c', 'd']
>>> things
['a', 'b', 'c', 'd']
>>> print(things[1])
b
>>> things[1] = 'z'
>>> print(things[1])
z
>>> things
['a', 'z', 'c', 'd']
```

```
>>> mix = [1, 'b', False, ["a", 1, 10.03, True]]
>>> type(mix)
<class 'list'>
>>> mix
[1, 'b', False, ['a', 1, 10.03, True]]
>>> type(mix[0])
<class 'int'>
>>> type(mix[1])
<class 'str'>
>>> type(mix[2])
<class 'bool'>
>>> type(mix[3])
<class 'list'>
```



ex39

```
>>> stuff = {'name': 'Zed', 'age': 39, 'height': 5 * 12 + 2}
>>> print(stuff['name'])
Zed
>>> print(stuff['age'])
39
>>> print(stuff['height'])
62
>>> stuff['city'] = "SF"
>>> print(stuff['city'])
SF
>>> stuff
{'name': 'Zed', 'age': 39, 'height': 62, 'city': 'SF'}
```

```
>>> stuff[1]
'Wow'
>>> stuff[1] = "Wow"
>>> stuff[1]
'Wow'
>>> stuff[2] = "Neato"
>>> stuff[2]
'Neato'
>>> stuff
{'name': 'Zed', 'age': 39, 'height': 62, 'city': 'SF', 1: 'Wow', 2: 'Neato'}
```

```
>>> stuff
{'name': 'Zed', 'age': 39, 'height': 62, 'city': 'SF', 1: 'Wow', 2: 'Neato'}
>>> del stuff['city']
>>> del stuff[1]
>>> del stuff[2]
>>> stuff
{'name': 'Zed', 'age': 39, 'height': 62}
```



## Ex 39

```
1  states = {
2    'oregon': 'OR',
3    'Florida': 'FL',
4    'California': 'CA',
5    'New York': 'NY',
6    'Michigan': 'MI'
7  }
8
9  cities = {
10   'CA': 'San Francisco',
11   'MI': 'Detroit',
12   'FL': 'Jacksonville'
13 }
14
15 # add some more cities
16 cities['NY'] = 'New York'
17 cities['OR'] = 'Portland'
18
19 print("cities: ", cities)
20 print("states: ", states)
21
```

```
cities: {'CA': 'San Francisco', 'MI': 'Detroit', 'FL': 'Jacksonville', 'NY': 'New York', 'OR': 'Portland'}
states: {'oregon': 'OR', 'Florida': 'FL', 'California': 'CA', 'New York': 'NY', 'Michigan': 'MI'}
```

## Ex39

```
22 # print out some cities
23 print('-' * 10)
24 print("print out some cities")
25 print("NY State has: ", cities['NY'])
26 print("OR State has: ", cities['OR'])
27
28 # Print some states
29 print('-' * 10)
30 print("print some states")
31 print("Michigan's abbreviation is:", states['Michigan'])
32 print("Florida's abbreviation is: ", states['Florida'])
33
34 # do it by using the state then cities Dictionaries
35 print("do it by using the state then cities Dictionaries")
36 print('-' * 10)
37 print("Michigan has: ", cities[states['Michigan']])
38 print("Florida has: ", cities[states['Florida']])
39
```

```
print out some cities
NY State has:  New York
OR State has:  Portland
-----
print some states
Michigan's abbreviation is: MI
Florida's abbreviation is:  FL
do it by using the state then cities Dictionaries
-----
Michigan has:  Detroit
Florida has:   Jacksonville
```

## Ex39

```
40 # print every state abbreviation
41 print('-' * 10)
42 print("every state abbreviation")
43 for state, abbrev in list(states.items()):
44     print(f"{state} is abbreviated {abbrev}")
45
46 # print every city in state
47 print('-' * 10)
48 print("print every city in state")
49 for abbrev, city in list(cities.items()):
50     print(f"{abbrev} has the city {city}")
51
52 # now do both at the same time
53 print('-' * 10)
54 print("every state abbreviation")
55 for state, abbrev in list(states.items()):
56     print(f"{state} state is abbreviated {abbrev}")
57     print(f"and has the city {cities[abbrev]}")
58
```

```
every state abbreviation
oregon is abbreviated OR
Florida is abbreviated FL
California is abbreviated CA
New York is abbreviated NY
Michigan is abbreviated MI
-----
print every city in state
CA has the city San Francisco
MI has the city Detroit
FL has the city Jacksonville
NY has the city New York
OR has the city Portland
-----
every state abbreviation
oregon state is abbreviated OR
and has the city Portland
Florida state is abbreviated FL
and has the city Jacksonville
California state is abbreviated CA
and has the city San Francisco
New York state is abbreviated NY
and has the city New York
Michigan state is abbreviated MI
and has the city Detroit
-----
```

## Ex 39

```
59 print('-' * 10)
60 print("safely get a abbreviation by state that might not be there")
61 state = states.get('Texas')
62
63 print("state.get: ", state, "and type: ", type(state))
64 if not state:
65     print("Sorry, no Texas.")
66
67 # get a city with a default value
68 city = cities.get('TX', 'Does not Exist')
69 print(f"The city for the state 'TX' is: ", {city})
70
71 # get a city with a default value
72 city = cities.get('CA', 'Does not Exist')
73 print(f"The city for the state 'CA' is: ", {city})
74
```

```
-----
safely get a abbreviation by state that might not be there
state.get:  None and type:  <class 'NoneType'>
Sorry, no Texas.
The city for the state 'TX' is:  {'Does not Exist'}
The city for the state 'CA' is:  {'San Francisco'}
```

## Quiz 2 (L3Q2)

Are dictionaries mutable?

Given `d = {'a': 1, 2: 'b'}`,

- What are the keys?
- what value does `d[2]` evaluate to?
- What value does `d['b']` evaluate to?
- What value does `d.get(2)` evaluate to?
- What value does `d.get('b')` evaluate to?
- What does `d` refer to after executing `d['w'] = 3`?
- What does `d` refer to when executing `d['w'] = 1` after the above?

```
>>> d = {'a':1, 2:'b'}
>>> d[2]
'b'
>>> d['b']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'b'
>>> d.get(2)
'b'
>>> d.get('b')
>>> d['w'] = 3
>>> d
{'a': 1, 2: 'b', 'w': 3}
>>> d['w'] = 1
>>> d
{'a': 1, 2: 'b', 'w': 1}
>>>
```



## Discussion 2

Operations on dictionaries

(<https://docs.python.org/3/library/stdtypes.html#typesmapping> )

If the variable `b` refers to `{"one": 1, "two": 2, "three": 3, "four": 4}`,

– what does each of the following evaluate to?

`len(b)`

`"one" in b`

`4 in b`

`b.items()`

`b.keys()`

`b.values()`

– What does `b` refer to after executing `del b["four"]`?

```
>>> b = {"one":1, "two":2, "three":3, "four": 4}
>>> len(b)
4
>>> "one" in b
True
>>> 4 in b
False
>>> b.items()
dict_items([('one', 1), ('two', 2), ('three', 3), ('four', 4)])
>>> b.keys()
dict_keys(['one', 'two', 'three', 'four'])
>>> b.values()
dict_values([1, 2, 3, 4])
>>>
>>> del b["four"]
>>> b
{'one': 1, 'two': 2, 'three': 3}
>>>
```

## 4. Sorting Dictionary



## Discussion 3 (L3D3)

Sorting dictionary on keys or values

<https://docs.python.org/3/library/collections.html#ordereddict-examples-and-recipes>

Given d = {'banana': 3, 'apple': 4, 'pear': 1, 'orange': 2}

- write code to generate a dictionary sorted by keys {'apple': 4, 'banana': 3, 'orange': 2, 'pear': 1}
- write code to generate a dictionary sorted by values {'pear': 1, 'orange': 2, 'banana': 3, 'apple': 4}

```
>>> d = {'banana':3, 'apple':4, 'pear':1, 'orange':2}
>>> d.keys
<built-in method keys of dict object at 0x000001B78F686E58>
>>> d.keys()
dict_keys(['banana', 'apple', 'pear', 'orange'])
>>> sorted(d.keys())
['apple', 'banana', 'orange', 'pear']
>>> sorted(d.keys(), reverse=False)
['apple', 'banana', 'orange', 'pear']
>>> sorted(d.keys(), reverse=True)
['pear', 'orange', 'banana', 'apple']
>>> d_k = {}
>>> for key in sorted(d.keys()):
...     print(key,d[key])
...     d_k[key] = d[key]
...
apple 4
banana 3
orange 2
pear 1
>>> print(d_k)
{'apple': 4, 'banana': 3, 'orange': 2, 'pear': 1}
>>>
```

```
1  d = {'banana':3, 'apple':4, 'pear':1, 'orange':2}
2
3  print("d.items(): ", d.items())
4  print("sorted items: ", sorted(d.items(),reverse=True))
5  d_i = {} # initialize the value
6  for key, item in sorted(d.items(),reverse=True):
7      d_i[key] =item
8
9  print("sorted dict: ", d_i)
10 |
```

```
PS D:\_0SUSS\ANL251Python\MyCode\L3> python L3D3b.py
d.items(): dict_items([('banana', 3), ('apple', 4), ('pear', 1), ('orange', 2)])
sorted items: [('pear', 1), ('orange', 2), ('banana', 3), ('apple', 4)]
sorted dict: {'pear': 1, 'orange': 2, 'banana': 3, 'apple': 4}
PS D:\_0SUSS\ANL251Python\MyCode\L3>
```

```
>>> a = dict(one=1, two=2, three=3)
>>> b = {'one': 1, 'two': 2, 'three': 3}
>>> c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
>>> d = dict([('two', 2), ('one', 1), ('three', 3)])
>>> e = dict({'three': 3, 'one': 1, 'two': 2})
>>>
>>> a == b == c == d == e
True
>>> a
{'one': 1, 'two': 2, 'three': 3}
>>> b
{'one': 1, 'two': 2, 'three': 3}
>>> c
{'one': 1, 'two': 2, 'three': 3}
>>> d
{'two': 2, 'one': 1, 'three': 3}
>>> e
{'three': 3, 'one': 1, 'two': 2}
>>>
```

```
>>> import collections
>>> a0 = collections.OrderedDict(one=1, two=2, three=3)
>>> a0 == a
True
>>> a0
OrderedDict([('one', 1), ('two', 2), ('three', 3)])
>>> a1 = collections.OrderedDict(three=3, one=1, two=2)
>>> a1
OrderedDict([('three', 3), ('one', 1), ('two', 2)])
>>> a1 == a0
False
>>> a1 == a
True
>>> a0 == e
True
>>> a1 == e
True
>>> a1 == d
True
>>> type(a0)
<class 'collections.OrderedDict'>
>>> type(a)
<class 'dict'>
>>>
```

```
1 import collections
2 d = {'banana':3, 'apple':4, 'pear':1, 'orange':2}
3
4 d_ordered = collections.OrderedDict()
5 for key, value in d.items():
6     print(key,value)
7     d_ordered[key] = value
8
9 print(d)
10 print(d_ordered)
```

```
banana 3
apple 4
pear 1
orange 2
{'banana': 3, 'apple': 4, 'pear': 1, 'orange': 2}
OrderedDict([('banana', 3), ('apple', 4), ('pear', 1), ('orange', 2)])
```

You may change the data type for the results.



# 5. Loop over Dictionary



## Quiz 3 (L3Q3)

If the variable `dishes` refers to `{'eggs': 2, 'sausage': 1, 'bacon': 1, 'spam': 500}`,

- write a program to count the total number of all the dishes.
- write a program to increase each value by 1.

```
2 dishes = {'eggs':2, 'sausage':1, 'bacon':1, 'spam':500}
3
4 total = 0
5 for key, value in dishes.items():
6     print(key, value)
7     total = total + int(value)
8     print("total= ", total)
9
10 print("Final total= ", total)
```

```
eggs 2
total= 2
sausage 1
total= 3
bacon 1
total= 4
spam 500
total= 504
Final total= 504
```

```
2 dishes = {'eggs':2, 'sausage':1, 'bacon':1, 'spam':500}
3
4 print("old dishes: ", dishes)
5 for key, value in dishes.items():
6     dishes[key] = value + 1
7
8 print("new dishes: ", dishes)
```

```
old dishes: {'eggs': 2, 'sausage': 1, 'bacon': 1, 'spam': 500}
new dishes: {'eggs': 3, 'sausage': 2, 'bacon': 2, 'spam': 501}
```

## Discussion 4 (L3D4)

Given `color_to_fruit = {'orange': 'orange', 'purple': 'plum', 'green': 'pear', 'yellow': 'banana', 'red': 'pomegranate'}`, write a program to add a new fruit (e.g. green watermelon), but not to replace the existing one (e.g. pear).

```
1 color_to_fruit = {'orange': 'orange', 'purple': 'plum',  
2 'green': 'pear', 'yellow': 'banana', 'red': 'pomegranate'}  
3  
4 print("old: ", color_to_fruit )  
5 color_to_fruit['green'] = 'watermelon'  
6 print("new: ", color_to_fruit )  
7  
8 print("-"*10)  
9 color_to_fruit = {'orange': 'orange', 'purple': 'plum',  
10 'green': 'pear', 'yellow': 'banana', 'red': 'pomegranate'}  
11 print("old: ", color_to_fruit )  
12 color_to_fruit['green'] = [color_to_fruit['green'], 'watermelon']  
13 print("new: ", color_to_fruit )  
14
```

```
old: {'orange': 'orange', 'purple': 'plum', 'green': 'pear', 'yellow': 'banana', 'red': 'pomegranate'}  
new: {'orange': 'orange', 'purple': 'plum', 'green': 'watermelon', 'yellow': 'banana', 'red': 'pomegranate'}  
-----  
old: {'orange': 'orange', 'purple': 'plum', 'green': 'pear', 'yellow': 'banana', 'red': 'pomegranate'}  
new: {'orange': 'orange', 'purple': 'plum', 'green': ['pear', 'watermelon'], 'yellow': 'banana', 'red': 'pomegranate'}
```

## Discussion 5 (L3D5)

Given `id_to_grade = {'1389': 55.0, '1377': 85.0, '1311': 77.5, '1078': 62.5, '0941': 55.0, '0052': 77.5}`, write a program to generate a dictionary where each key is a grade and each value is the list of ids of students who earned that grade.

```
2 id_to_grade = {'1389':55.0, '1377':85.0,  
3 '1311':77.5,'1078':62.5,'0941':55.0,'0052':77.5, '1400':55.0}  
4  
5 # print("dict")  
6 grade_id = {}  
7 for key, value in id_to_grade.items():  
8     exist = grade_id.get(value)  
9     if not exist:  
10         grade_id[value] = [key]  
11     else:  
12         #grade_id[value] = [grade_id[value], key] # nested list  
13         grade_id[value].append(key)  
14  
15 print("old: ", id_to_grade)  
16 print("new ", grade_id)
```

```
old: {'1389': 55.0, '1377': 85.0, '1311': 77.5, '1078': 62.5, '0941': 55.0, '0052': 77.5, '1400': 55.0}  
new {55.0: ['1389', '0941', '1400'], 85.0: ['1377'], 77.5: ['1311', '0052'], 62.5: ['1078']}
```

Q&A  
Thank you