

A dictionary can be used to store data. Whereas a list stores an arbitrary collection of items, a *dictionary stores key-value pairs*. The running example here is a dictionary that maps stock ticker symbols to prices. The ticker symbol is the key, the price is the value.

- Keys *must* be unique. E.g., two stocks cannot have the same ticker symbol.
- Keys *must* be an immutable type. E.g., int, float, str but not list.
- Values are not necessarily unique. E.g., two stocks might have the same price.

To create an empty dictionary, use curly braces.

```
>>> stocks = {}      # curly braces create an empty dictionary
```

To initialize a dictionary with a set of key value pairs, the syntax is this.

```
>>> stocks = { 'FB':20, 'MSFT':28, 'GOOG':652, 'AAPL':537 } # initialize w/ 4 stocks
>>> stocks
{'GOOG': 652, 'FB': 20, 'AAPL': 537, 'MSFT': 28}
```

Notice how python does not preserve the order of the stocks, nor are they sorted alphabetically. Unlike lists and strings, dictionaries are not ordered.

Getting and modifying values

```
>>> stocks['FB']          # get value associated with key
20
>>> stocks['ZNGA'] = 2    # add new key, value pair
>>> stocks['ZNGA']
2
>>> stocks                # can see that ZNGA has been added
{'GOOG': 652, 'ZNGA': 2, 'FB': 20, 'AAPL': 537, 'MSFT': 28}
>>> stocks['ZNGA'] = 3    # replace value associated with key
>>> stocks['ZNGA']
3
>>> stocks                # still only one ZNGA, now with new price
{'GOOG': 652, 'ZNGA': 3, 'FB': 20, 'AAPL': 537, 'MSFT': 28}
>>> stocks['FB'] = stocks['FB'] + 9 # update value associated with key
>>> stocks['FB']
29
```

Operators and methods

```
>>> len(stocks)          # returns number of key, value pairs
5
>>> 'GOOG' in stocks     # can use in operator, just like lists
True
>>> 'GOOGLE' in stocks
False
>>> 652 in stocks        # in operator only checks keys, not values
```

```

False
>>> stocks.pop('ZNGA')           # remove key, value pair
3
>>> stocks
{'GOOG': 652, 'FB': 29, 'AAPL': 537, 'MSFT': 28}
>>> stocks.keys()                # returns list of keys
['GOOG', 'FB', 'AAPL', 'MSFT']
>>> stocks.values()              # returns list of values
[652, 29, 537, 28]
>>> stocks.items()               # returns list of (key, value) tuples
[('GOOG', 652), ('FB', 29), ('AAPL', 537), ('MSFT', 28)]

```

Iteration

You can loop over the keys of a dictionary, like this.

```

>>> stocks = { 'FB':20, 'MSFT':28, 'GOOG':652}
>>> for k in stocks:
...     print k, # print just the key (ticker symbol)
GOOG FB MSFT
>>> for k in stocks:
...     print stocks[k], # print just the value (stock price)
652 20 28
>>> for k in stocks:
...     print k, stocks[k] # print both key (ticker symbol) and value (stock price)
GOOG 652
FB 20
MSFT 28

```

You can also loop over the lists returned by the keys, values, and items methods.

Exercises

Some solutions are presented in class and also included in the moodle version of this handout.

1. Given a list of ticker symbols, add each ticker symbol to stocks with a price of 10.

Solution:

```

stocks = { 'FB':20, 'MSFT':28, 'GOOG':652, 'AAPL':537 }
tickers = ['AMZN', 'NFLX']

for symbol in tickers:
    stocks[symbol] = 10

```

- Increase the price of 'GOOG' price by 10%. However, if 'GOOG' is not in the dictionary, add it with a price of 1000.

Solution:

```
stocks = { 'FB':20, 'MSFT':28, 'GOOG':652, 'AAPL':537 }
```

write code to increase the price of 'GOOG' price by 10%.
however, if 'GOOG' not in stocks, add it with a price of
1000.

```
if 'GOOG' not in stocks:
    stocks['GOOG'] = 1000
else:
    stocks['GOOG'] = stocks['GOOG'] * 1.10
```

- Write a function `max_price` that takes a dictionary of stocks and returns the highest stock price.

Solution:

```
def max_price(stocks):
    '''(dict of {str:int}) -> int
    Given dictionary that maps stocks symbols to price, returns
    highest stock price.
    >>> stocks = { 'GOOG':652, 'FB':20, 'AAPL':537, 'MSFT':26 }
    >>> max_price(stocks)
    652
    '''
    return max(stocks.values())
```

- Write a function `priciest_stock` that takes a dictionary of stocks and returns the ticker symbol of the stock with the highest stock price.

Solution:

```
def priciest_stock(stocks):
    '''(dict of {str:int}) -> str
    Given dictionary that maps stocks symbols to price, returns
    the ticker symbol of the stock with the highest stock price.
    >>> stocks = { 'GOOG':652, 'FB':20, 'AAPL':537, 'MSFT':26 }
    >>> priciest_stock(stocks)
    'GOOG'
```

```
'''
max_price = -1
priciest = None
for ticker in stocks:
    price = stocks[ticker]
    if price > max_price:
        max_price = price
        priciest = ticker
return priciest
```

5. Write a function `pricey_stocks` that takes a dictionary and a cutoff (an int) and returns a list of ticker symbols whose price is above the cutoff.

Solution:

```
def pricey_stocks(stocks, cutoff):
    '''(dict of {str:int}, int) -> list of str
    Given dictionary that maps stocks symbols
    to price, returns list of stocks whose price
    is above cutoff.
    >>> stocks = { 'GOOG':652, 'FB':20, 'AAPL':537, 'MSFT':26 }
    >>> pricey_stocks(stocks, 500)
    ['GOOG', 'AAPL']
    '''
    pricey = []
    for ticker in stocks:
        if stocks[ticker] > cutoff:
            pricey.append(ticker)
    return pricey
```