```
# -*- coding: utf-8 -*-
"""
Created on Sat May 19 21:00:28 2018

@author: Tawanda Vera
"""
"""
Q2. Develop a simple model to estimate execution slippage of each of the order
types based on latest available values of the 9 main data points:
• Open,
• High,
• Low,
• Close,
• Volume,
• Best Bid,
• Best Ask,
• Highest Bid Quantity,
• Highest Ask Quantity.

In Other words, the model should provide a realistic analysis of how much
slippage we can expect in an order given the latest available values of the 9
data points.
• Write a simple Python program that implements your model. The program should
be able printout the expected slippages in the different order types based on
values of 9 data points (Do not ask the user to input these values manually,
rather define them as variables in your python file with proper commenting)
"""

def calc_slippage(book, volume, side):
    """Calculate slippage given a market order and order book snapshot.
    Args:
        book: A dict representing a snapshot of an order book.
            Contains two dicts for each side of the order book.
            Each dict representing order book side has a price level as
            key and volume at price level as value.
                Example: book = {
                                'bid': {100: 5, 95: 2},
                                'ask': {120: 3, 115: 4}
                                }
            This book has two bid price levels: 100 with volume of 5 and
            95 with volume of 2; Also two ask levels: 120 with volume 3 and
            115 with volume 4.
            Volume: Volume of a market order that is used to calculate
            corresponding slippage.
            Side: Side of a market order; either 'bid' or 'ask'.
    Returns:
    A dict with slippage cost and percentage, and some other intermediate
    data that was used in the calculation:
    slippage_cost - absolute value of slippage cost for a given market order
    slippage_frac - what fraction of the quote total price does the slippage
    cost constitute
    quote_price - either lowest ask or highest bid, depending on the selected
    side of the market
    quote_total - total cost of market order when entire volume is filled at
    quote price
    actual_total - total cost of market order when order is filled using
    available order book liquidity
    """
    price_levels = get_price_levels(book, side)

    # Assume the high value is the maximum in 'ask'
    High =  max(book['ask'])

    # Assume the low value is the minimum in 'bid'
    Low = min(book['bid'])

    # Assume the first ask price in book is the open
    Open = list(book['ask'])[0]

    # Assume the last ask price in book is the close
    Close = list(book['ask'])[-1]

    # identical to get_quote_price, just don't need to recalc price_levels twice
    quote_price = price_levels[0][0]
    quote_total = volume * quote_price
    actual_total = 0
    volume_to_fill = volume
    for price, level_volume in price_levels:
        if volume_to_fill - level_volume <= 0:
            actual_total += volume_to_fill * price
            break
        volume_to_fill -= level_volume
        actual_total += level_volume * price

    # Bid-Ask Spread
    c0 = Close - Open
    c1 = High - Low
    bid_ask_spread = c0 + c1
    slippage_cost = abs(quote_total - actual_total + bid_ask_spread)
    slippage_frac = slippage_cost / quote_total
    slippage = {'slippage_cost': slippage_cost,
                'slippage_frac': slippage_frac,
                'quote_price': quote_price,
                'quote_total': quote_total,
                'actual_total': actual_total,
                'high' : High,
                'low' : Low,
                'open': Open,
                'close': Close
                }
    return slippage

def get_quote_price(book, side):
    """Useful if need just the current quote price.
    Args:
        book: Same dict as in book arg to calc_slipage().
        side: Either 'bid' or 'ask'.
    Returns:
        The current quote price at the chosen side of the market, i.e.
        highest bid(Best Bid) or lowest ask(Best Ask).
    """
    price_levels = get_price_levels(book, side)
    quote_price = price_levels[0][0]
    return quote_price
```

```python
def get_price_levels(book, side):
    """Returns price levels according to their fill order.
    Args:
        book: Same dict as in book arg to calc_slipage().
        side: Either 'bid' or 'ask'.
    Returns:
        A list of price levels from a chosen side of the order book sorted in the following order:
            Fill order for bid side: highest price -> lowest price
            Fill order for ask side: lowest price -> highest price
        Each price level item in the list is a tuple: (price, volume).
        * Sorting tuples without specifying a key works fine here since tuples are compared
          position by position, and the first element of price level tuple is the price.
    """
    if side == 'bid':
        levels = book['bid']
        price_levels = sorted(levels.items(), reverse=True)
    elif side == 'ask':
        levels = book['ask']
        price_levels = sorted(levels.items())
    return price_levels

if __name__=='__main__':
    # test calculation using an imaginery index
    book = {'bid': {8962.9: 5.86106893, 8962.2: 0.49877177,
                    8961: 0.49810183, 8960.1: 0.40911833,
                    8960: 0.55656053, 8959.8: 0.13452915,
                    8959.6: 0.1, 8958.4: 0.5001, 8958.3: 0.9174,
                    8958.1: 1, 8958: 8.29, 8957.8: 0.0050135,
                    8957.7: 0.49948499, 8957.5: 0.002, 8956.6: 0.1,
                    8954.8: 0.28966197, 8954.6: 0.5633873,
                    8954.5: 0.50004267, 8954: 1.499, 8953.8: 0.13452915,
                    8953.5: 0.49848494, 8953: 8.24, 8951.9: 0.49831686,
                    8950.4: 2.5, 8950.1: 3},
        'ask': {8963: 1.7092689, 8964: 7.41318758, 8965.6: 0.41137858,
                8967: 1.1143, 8967.7: 1.1142, 8967.9: 1.715, 8968: 0.12633553,
                8968.9: 1.1137, 8969: 0.025, 8970: 1.25, 8970.1: 0.45,
                8973.3: 4.61, 8973.4: 0.45, 8975.4: 0.591, 8976.6: 0.07381446,
                8978.5: 0.11288748, 8979: 1.05179565, 8979.1: 0.01327321,
                8980: 5.75, 8980.2: 0.02, 8981.6: 0.01179565, 8981.9: 1,
                8982: 8, 8982.1: 0.01327321, 8982.4: 0.2}}
    slippage = calc_slippage(book, 15, 'ask')
    print(slippage)

"""
{'slippage_cost': 15.963874591967397,
'slippage_frac': 0.00011873907242342517,
'quote_price': 8963, 'quote_total': 134445,
 'actual_total': 134480.73612540803,
 'high': 8982.4, 'low': 8950.1, 'open': 8963, 'close': 8982.4}
"""
```