# UROPS Project Presentation 4

Wang Zexin

Chapter 15 Valuation Framework
of Python for Finance

*Quantitative Finance*
*National University of Singapore*

February 6, 2017

# Today's Agenda

1. Valuation Framework
   - Risk-Neutral Discounting
   - Market environment
   - Wrapper class

# Changes due to different Python version

We are using Python 3.6 while the version in the book is Python 2.7
So here is a list of items to change

- print x now becomes print(x)
- dict.iteritems() now becomes dict.items()
- xrange now becomes range
- lambda (k, v) : (v, k) is no longer available
- instead we can only use: lambda x : (x[1], x[0])
- x / 2 is float division, while x // 2 is integer division

# Valuation Framework

- Risk-neutral Discounting
- Market environment
- Wrapper class

# Modelling and Handling of dates

This function takes in a list of concrete dates and convert to year fractions for obtaining accurate discounting purposes. (assuming one year has 365 days)

```
def get_year_deltas(date_list, day_count=365.):
    ''' Return vector of floats with day deltas in years.
    Parameters
    ==========
    date_list : list or array of datetime objects
    day_count : float number of days for a year
        (to account for different conventions)
    delta_list : array of year fractions
    '''
    start = date_list[0]
    delta_list = [(date - start).days / day_count
                  for date in date_list]
    return np.array(delta_list)
```

# Modelling and Handling of dates

An example of using this:

```
>>> import datetime as dt
>>> dates = [dt.datetime(2015,1,1),
        dt.datetime(2015,7,1), dt.datetime(2016,1,1)]
>>> get_year_deltas(dates)
array([ 0.        ,  0.49589041,  1.        ])
```

# Constant short rate

```
from get_year_deltas import *
class constant_short_rate(object):
    '''name : string
    short_rate : float (positive)
    constant rate for discounting
    get_discount_factors :
    get discount factors given list of datetime or year fractions
    '''
    def __init__(self, name, short_rate):
        self.name = name
        self.short_rate = short_rate
        if short_rate < 0:
            raise ValueError('Short rate negative.')
    def get_discount_factors(self, date_list, dtobjects=True):
        if dtobjects is True:
            dlist = get_year_deltas(date_list)
        else:
            dlist = np.array(date_list)
        dflist = np.exp(self.short_rate * np.sort(-dlist))
        return np.array((date_list, dflist)).T
```

This is a class of constant short rate build only on name and short rate. Possible extension: maybe we can include the discounting/compounding mechanism as well?

# Constant short rate

```
>>> import datetime as dt
>>> dates = [dt.datetime(2015,1,1),
        dt.datetime(2015,7,1), dt.datetime(2016,1,1)]
>>> csr = constant_short_rate('csr', 0.05)
>>> csr.get_discount_factors(dates)
array([[datetime.datetime(2015, 1, 1, 0, 0), 0.951229424500714],
       [datetime.datetime(2015, 7, 1, 0, 0), 0.975510338765722 8],
       [datetime.datetime(2016, 1, 1, 0, 0), 1.0]], dtype=object)
```

```
>>> deltas = get_year_deltas(dates)
>>> csr.get_discount_factors(deltas, dtobjects=False)
array([[ 0.        ,  0.95122942],
       [ 0.49589041,  0.97551034],
       [ 1.        ,  1.        ]])
```

The return value of *get_discount_factor* is a list of pairs of *datetime* object and corresponding discount factors.

# Market environment

The advantage of using a class as the market environment

- Convenience for consistent modelling
- Usage of dictionaries to store constants, lists and curves
- Ease of interactions between environments and other objects
- Act as a point of integration between previous functions and classes and further developed financial models

# Market environment - implementation

```
from constant_short_rate import *
import get_year_deltas

class market_environment(object):
    '''name: string name of the market environment
    pricing_date : datetime object date of the market environment
    add_constant : adds a constant (e.g. model parameter)
    get_constant : gets a constant
    add_list : adds a list (e.g. underlyings)
    get_list : gets a list
    add_curve : adds a market curve (e.g. yield curve)
    get_curve : gets a market curve
    add_environment : adds and overwrites whole market environments
    with constants, lists, and curves'''
    def __init__(self, name, pricing_date):
        self.name = name
        self.pricing_date = pricing_date
        self.constants = {}
        self.lists = {}
        self.curves = {}
```

# Market environment - implementation

```
def add_constant(self, key, constant):
    self.constants[key] = constant
def get_constant(self, key):
    return self.constants[key]
def add_list(self, key, list_object):
    self.lists[key] = list_object
def get_list(self, key):
    return self.lists[key]
def add_curve(self, key, curve):
    self.curves[key] = curve
def get_curve(self, key):
    return self.curves[key]
def add_environment(self, env):
    # overwrites existing values, if they exist
    for key in env.constants:
        self.constants[key] = env.constants[key]
    for key in env.lists:
        self.lists[key] = env.lists[key]
    for key in env.curves:
        self.curves[key] = env.curves[key]
```

# Market environment - testing

```
>>> from market_environment import *
>>> import datetime as dt
>>> dates = [dt.datetime(2015, 1, 1), dt.datetime(2015, 7, 1),
             dt.datetime(2016, 1, 1)]
>>> csr = constant_short_rate('csr', 0.05)
>>> me_1 = market_environment('me_1', dt.datetime(2015,1,1))
>>> me_1.add_list('symbols', ['AAPL', 'MSFT', 'FB'])
>>> me_1.get_list('symbols')
['AAPL', 'MSFT', 'FB']
>>> me_2 = market_environment('me_2', dt.datetime(2015,1,1))
>>> me_2.add_constant('volatility', 0.2)
>>> me_2.add_curve('short_rate', csr)
>>> me_2.get_curve('short_rate')
<constant_short_rate.constant_short_rate object at 0x0000000002EC09E8>
```

The market environment is able to:

- Add/Retrieve list of symbols
- Add/Retrieve/Update parameters for the model
- Add/Retrieve constant short rate

# Market environment - testing

```
>>> me_1.add_environment(me_2)
>>> me_1.get_curve('short_rate')
<constant_short_rate.constant_short_rate object at 0x0000000002EC
>>> me_1.constants
{'volatility': 0.2}
>>> me_1.lists
{'symbols': ['AAPL', 'MSFT', 'FB']}
>>> me_1.curves
{'short_rate': <constant_short_rate.constant_short_rate object at
>>> me_1.get_curve('short_rate').short_rate
0.05
```

The market environment is able to:

- Add all the attributes of another market environment

# Wrapper class - implementation

```
import datetime as dt

from get_year_deltas import get_year_deltas
from constant_short_rate import constant_short_rate
from market_environment import market_environment
```

With this *dx_frame.py*, we are now able to import the three components in one line.

# Wrapper class - testing

```
>>> from dx_frame import *
>>> market_environment
<class 'market_environment.market_environment'>
>>> constant_short_rate
<class 'constant_short_rate.constant_short_rate'>
>>> get_year_deltas
<function get_year_deltas at 0x0000000002E16840>
```

We are now able to import the entire three components in one line

# Wrapper class - testing

If we add an *__init__.py* which has exactly the same content as *dx_frame.py* in the same directory, we should be able to directly import like this.

```
>>> import dx_frame
>>> dx_frame.get_year_deltas
<function get_year_deltas at 0x0000000002C26840>
>>> dx_frame.constant_short_rate
<class 'constant_short_rate.constant_short_rate'>
>>> dx_frame.market_environment
<class 'market_environment.market_environment'>
```

# Thank You

*E0007424@u.nus.edu*