Prof. Dr. Natalie Packham
LV-Nr. 425093

Summer term 2019

Computational Finance and FinTech – Problem Set 2 with solutions

Exercise 1. Write a programm that finds all numbers that are divisible by 7 but are not multiples of 5 betweeen 2000 and 3200 (both included). The numbers obtained should be printed in a comma-separated sequence on a single line.

Solution 1.

```
l=[]
for i in range(2000,3201):
    if (i%7==0) and (i%5!=0):
        l.append(str(i))

print(", ".join(l))
```

Exercise 2. Write a function that computes the present value of a growing annuity:

$$\text{PV} = \frac{C}{r-g}\left(1 - \frac{(1+g)^n}{(1+r)^n}\right),$$

where r is the discount rate, g is the growth rate, C is the amount paid and n is the number of periods (years).
Use the function to calculate the PV of an annuity that pays 100 annually for ten years using a discount rate of 5%.

Solution 2.

```
def annuity(r, g, C, n):
    return C/(r-g) * (1 - (1+g)**n / (1+r)**n)
```

```
annuity(0.05, 0, 100,10)
```

Output:

```
1772.1734929184818
```

Exercise 3. Using the function from the previous exercise, build a mortgage calculator that prints out the monthly mortgage payment given an annual percentage rate (note: $r = \text{APR}/12$), the loan amount and the number of years (note: $n = \text{years} \cdot 12$).
Use the function to calculate the monthly payment on a loan of €100,000 at an APR of 2.5%. Verify that the PV is indeed €100,000 by calculating the sum of the discounted loan payments.

Solution 3.

```
def mortgage(r, g, N, n):
    return N / annuity(r/12,0,1,12*n)
```

```
mortgage(0.025, 0, 100000, 10)
```

Output:

```
942.6990170396044
```

```
pv=0;
for i in range(1,12*10+1):
    pv += 942.69901704/(1+0.025/12)**i

print(pv)
```

Output:

```
100000.00000003698
```

Exercise 4. Write a Python program to combine two dictionares adding values for common keys.
Example: Let

```
d1 = {'a': 100, 'b': 200, 'c':300}
d2 = {'a': 300, 'b': 200, 'd':400}
```

Sample output:

```
{'a': 400, 'b': 400, 'd': 400, 'c': 300}
```

Solution 4.

```
for key in set(d2.keys()).intersection(set(d1.keys())):
    d.update({key: (d2.get(key) + d1.get(key))})

print(d)
```

Output:

```
{'a': 400, 'b': 400, 'c': 300, 'd': 400}
```

Exercise 5. Generate a random walk path starting in 0 by accumulating normally distributed random numbers. Use a `Data Frame` object indexed by days (365 days) to accomondate the path. Plot the result.
Extend the `DataFrame` of random numbers to accomodate more paths and fill it with 5 paths.
Find out what the seed of a random number generator is and how you can set the seed in Python. Why would this be useful?
Hint: Use `numpy.random`.

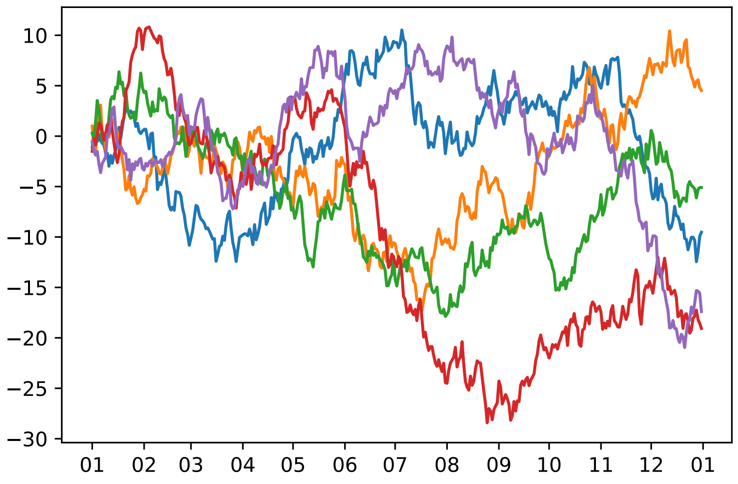
Solution 5.

```
import numpy as np
import pandas as pd
import matplotlib.dates as mdates
from pylab import plt, mpl # imports for visualisation

np.random.seed(123);
df=pd.DataFrame(np.random.normal(0,1,(365,5))).cumsum();
dates = pd.date_range('2019-1-1', periods = 365, freq = 'D')
df.index=dates

fig, ax = plt.subplots()
ax.xaxis.set_major_formatter(mdates.DateFormatter('%m'))
ax.xaxis.set_major_locator(mdates.MonthLocator())
ax.plot(df)
plt.show()
```

The seed sets the initial state in the pseudo-random-number generator ensuring that it will generate the same sequence of numbers over and over again. This is useful for example at the development stage, when you want to see only the changes in output from making changes in the code.
Output:


10
5
0
−5
−10
−15
−20
−25
−30
01
02
03
04
05
06
07
08
09
10
11
12
01