from xbbg import blp

import polars as pl

import datetime

class Main():

def \_\_init\_\_(self) -> None:

t = Tickers()

self.b = Bbg\_functions(t.tickers\_list)

Writer(self.b.df)

class Tickers():

PATH\_EXCEL = r"M:\BOLSA\MATHEUS\GerencialFundSec V2.2.xlsm"

COL\_TICKERS = "Sec\_bloom"

def \_\_init\_\_(self) -> None:

self.obtain\_df()

self.df\_to\_tickers()

def obtain\_df(self):

self.tickers\_df = pl.read\_excel(

self.PATH\_EXCEL, sheet\_name="Data\_Sec\_Eq",

read\_csv\_options={"skip\_rows": 2, "columns": [1]}

)

def df\_to\_tickers(self):

self.tickers\_list = self.tickers\_df.select(

pl.col(self.COL\_TICKERS)

).drop\_nulls()[:, 0].to\_list()

class Bbg\_functions():

def \_\_init\_\_(self, tickers) -> None:

self.tickers\_list = tickers

self.today = datetime.date.today()

self.filter\_by\_ex\_date()

self.obtain\_dividends()

def filter\_by\_ex\_date(self):

ex\_date = blp.bdp(tickers=self.tickers\_list, flds="DVD\_EX\_dt")

right\_ex\_date = blp.bdp(

tickers=self.tickers\_list, flds="EQY\_DVD\_RIGHT\_EX\_DT\_NEXT"

)

dividends\_today = pl.col("dvd\_ex\_dt") == self.today

ex\_date = pl.from\_pandas(

ex\_date.reset\_index()

).filter(dividends\_today)

rights\_today = pl.col("eqy\_dvd\_right\_ex\_dt\_next") == self.today

right\_ex\_date = pl.from\_pandas(

right\_ex\_date.reset\_index()

).filter(rights\_today)

self.ex\_date\_list = ex\_date.select(

pl.col("index")

).drop\_nulls()[:, 0].to\_list()

def obtain\_dividends(self):

today\_string = self.today.strftime("%Y%m%d")

self.df = blp.bds(

tickers=self.ex\_date\_list, flds="DVD\_HIST",

DVD\_START\_DT=today\_string, DVD\_END\_DT=today\_string

)

self.df = pl.from\_pandas(self.df.reset\_index())

class Writer():

OUTPUT\_PATH = r"M:\BOLSA\Victor\Routines\_files\Dividends\dividends"

def \_\_init\_\_(self, dataframe) -> None:

self.df = dataframe

self.format\_data()

self.write\_output()

def format\_data(self):

self.df = self.df.select(

pl.col(["index","dividend\_amount"])

).groupby("index").agg(

pl.col("dividend\_amount").sum()

)

def write\_output(self):

today\_string = datetime.datetime.today().strftime("%y-%m-%d")

self.df.write\_parquet(self.OUTPUT\_PATH + today\_string + ".parquet")

self.df.write\_csv(self.OUTPUT\_PATH + today\_string + ".txt", separator=";")

m = Main()

#a = blp.bds( tickers=m.b.tickers\_list, flds="EQY\_DVD\_ADJUST\_FACT", )

m.b.ex\_date\_list

m.b.tickers\_list

blp.bds(

tickers="JALL3 BZ EQUITY", flds="DVD\_HIST\_all",

)

########atentar para trades ate as 11

import datetime

import polars as pl

import requests

import zipfile

import base64

from io import BytesIO

class Main():

def \_\_init\_\_(self):

pass

class Dates():

def \_\_init\_\_(self) -> None:

self.today = datetime.datetime.now()

self.get\_last\_business\_day()

def get\_last\_business\_day():

PATH\_BROKERS = r"M:\BOLSA\Victor\Bases\Broker\_codes\broker\_codes.parquet"

URL\_B3 = "https://arquivos.b3.com.br/apinegocios/tickercsvbtb/viia3/2023-08-17"

def get\_broker\_codes(self):

self.brokers = pl.read\_parquet(self.PATH\_BROKERS)

def download\_from\_b3(self):

r = requests.get()

if r.status\_code == 200:

pass

else:

print("Problema em baixar o arquivo")

raise(Exception)

def unzip(self):

z = zipfile.ZipFile(BytesIO(r.content))

def create\_dataframe(self):

df = pl.read\_csv(z.open(z.namelist()[0]), separator=";")

def

col = df.columns

columns\_rename = {

"TaxaDeJurosDoTermoDoNegocio": "rate", "QuantidadeNegociada": "qty",

"HoraEntrada": "hour", "CodigoParticipanteDoador": "lender",

"CodigoParticipanteTomador": "borrower",

}

df = df.rename(columns\_rename)

df = df.select(pl.col(columns\_rename.values()))

df = df.with\_columns(

pl.col("rate").str.replace(",", ".").cast(pl.Float64),

pl.col("lender").cast(pl.Int32),

pl.col("borrower").cast(pl.Int32),

pl.when(pl.col("hour") < 1.09e8)

.then(pl.lit("janela"))

.otherwise(pl.lit("dia"))

.alias("hour")

)

functions = (pl.col("qty").sum().alias("traded"),

((pl.col("rate")\*pl.col("qty")).sum()/pl.col("qty").sum()).alias("average"),

pl.col("rate").min().alias("min"),

pl.col("rate").max().alias("max"),

(((pl.col("qty")

\*((pl.col("rate")

- ((pl.col("rate")\*pl.col("qty")).sum()/pl.col("qty").sum()))\*\*2)).sum()

/pl.col("qty").sum())\*\*0.5).alias("std"),)

top\_lender = df.groupby(["lender"]).agg(a).sort(by="traded", descending=True)

top\_lender = df.groupby(["lender"]).agg(

pl.col("qty").sum().alias("traded"),

((pl.col("rate")\*pl.col("qty")).sum()/pl.col("qty").sum()).alias("average"),

pl.col("rate").min().alias("min"),

pl.col("rate").max().alias("max"),

(((pl.col("qty")

\*((pl.col("rate")

- ((pl.col("rate")\*pl.col("qty")).sum()/pl.col("qty").sum()))\*\*2)).sum()

/pl.col("qty").sum())\*\*0.5).alias("std"),

).sort(by="traded", descending=True)

top\_lender[:5].join(brokers, left\_on="lender", right\_on="code")

top\_borrower[:5].join(brokers, left\_on="borrower", right\_on="code")

df

df.select(a)

m = Main()

import polars as pl

import xlwings as xw

class Main():

def \_\_init\_\_(self):

Inputs()

class Inputs():

WB\_NAME = "Inputs.xlsx"

SHEET\_NAME = "Last price"

RANGE = "G1:G"

PATH\_INDEXES = r"M:\BOLSA\Victor\Routines\_files\Download\_indexes"

PATH\_PRICES = r"M:\BOLSA\Victor\Routines\_files\Last\_prices"

def \_\_init\_\_(self):

self.obtain\_wb()

self.last\_line = 2

self.get\_sql\_info()

self.get\_indexes()

while True:

self.obtain\_prices()

def obtain\_wb(self):

self.wb\_inputs = ""

while self.wb\_inputs == "":

try:

app = xw.apps.active

self.wb\_inputs = app.books(self.WB\_NAME)

except KeyError:

print("Foco na intância errada, mude para input.xlsx")

self.sheet\_price = self.wb\_inputs.sheets[self.SHEET\_NAME]

app = xw.apps.active

self.wb\_gerencial = ""

while self.wb\_gerencial == "":

try:

app = xw.apps.active

self.wb\_gerencial = app.books("output.xlsx")

except KeyError:

print("Foco na intância errada, mude para output.xlsx")

self.sheet\_gerencial = self.wb\_gerencial.sheets["Sheet1"]

def get\_sql\_info(self):

last\_line\_sql = self.sheet\_price.range("T2").value

self.sheet\_sql = self.wb\_inputs.sheets["Base SQL"]

self.sql\_base = self.sheet\_sql.range("A1:G" + str(last\_line\_sql))

def get\_indexes(self):

today = datetime.datetime.today().strftime("%Y-%m-%d")

path = self.PATH\_INDEXES + "//" + today

self.ibov = pl.read\_parquet(path + "IBOV.parquet")

self.small = pl.read\_parquet(path + "SMLL.parquet")

self.ibx = pl.read\_parquet(path + "IBXX.parquet")

def obtain\_prices(self):

self.prices = self.sheet\_price.range(

self.RANGE + str(self.last\_line)

).value

self.sheet\_gerencial.range(

self.RANGE + str(self.last\_line)

).options(transpose=True).value = self.prices

self.last\_line = int(self.prices[0])

m = Main()

import polars as pl

import xlwings as xw

class Main():

def \_\_init\_\_(self):

Inputs()

class Inputs():

WB\_NAME = "Inputs.xlsx"

SHEET\_NAME = "Last price"

RANGE = "G1:G"

PATH\_INDEXES = r"M:\BOLSA\Victor\Routines\_files\Download\_indexes"

PATH\_PRICES = r"M:\BOLSA\Victor\Routines\_files\Last\_prices"

def \_\_init\_\_(self):

self.obtain\_wb()

self.last\_line = 2

self.get\_sql\_info()

self.get\_indexes()

while True:

self.obtain\_prices()

def obtain\_wb(self):

self.wb\_inputs = ""

while self.wb\_inputs == "":

try:

app = xw.apps.active

self.wb\_inputs = app.books(self.WB\_NAME)

except KeyError:

print("Foco na intância errada, mude para input.xlsx")

self.sheet\_price = self.wb\_inputs.sheets[self.SHEET\_NAME]

app = xw.apps.active

self.wb\_gerencial = ""

while self.wb\_gerencial == "":

try:

app = xw.apps.active

self.wb\_gerencial = app.books("output.xlsx")

except KeyError:

print("Foco na intância errada, mude para output.xlsx")

self.sheet\_gerencial = self.wb\_gerencial.sheets["Sheet1"]

def get\_sql\_info(self):

last\_line\_sql = self.sheet\_price.range("T2").value

self.sheet\_sql = self.wb\_inputs.sheets["Base SQL"]

self.sql\_base = self.sheet\_sql.range("A1:G" + str(last\_line\_sql))

def get\_indexes(self):

today = datetime.datetime.today().strftime("%Y-%m-%d")

path = self.PATH\_INDEXES + "//" + today

self.ibov = pl.read\_parquet(path + "IBOV.parquet")

self.small = pl.read\_parquet(path + "SMLL.parquet")

self.ibx = pl.read\_parquet(path + "IBXX.parquet")

def obtain\_prices(self):

self.prices = self.sheet\_price.range(

self.RANGE + str(self.last\_line)

).value

self.sheet\_gerencial.range(

self.RANGE + str(self.last\_line)

).options(transpose=True).value = self.prices

self.last\_line = int(self.prices[0])

m = Main()

import polars as pl

from numpy import busday\_count

from datetime import datetime

from datetime import time

from bcb import sgs

import requests

import re

from scipy.interpolate import CubicSpline

import xlwings as xw

class Main():

def \_\_init\_\_(self):

self.inputs = Inputs()

self.call\_correct\_class()

# self.test\_values()

def call\_correct\_class(self):

if self.inputs.index == "PRÉ":

self.lf = Fixed\_rate(self.inputs)

if self.inputs.index == "CDI +":

self.lf = Cdi\_spread(self.inputs)

if self.inputs.index == "CDI %":

self.lf = Cdi\_percent(self.inputs)

def test\_values(self):

b = Yield\_curve()

class Inputs():

WB\_NAME = "infos credito privado.xlsm"

SHEET\_NAME = "PU LF"

RANGE = "C2:C10"

def \_\_init\_\_(self):

#self.obtain\_inputs()

self.obtain\_inputs2()

self.adjust\_dates()

self.check\_liquidation()

def obtain\_inputs2(self):

self.index = "CDI +"

self.issue\_date = "24/05/2021"

self.maturity\_date = "24/05/2024"

self.liquidation\_date = "15/08/2023"

self.issue\_rate = 0.0117

self.deal\_rate = 0.011

self.issuance\_par\_value = 1000

self.last\_coupon = "24/05/2023"

self.coupon\_frequency = 6

def obtain\_inputs(self):

self.wb = xw.Book(self.WB\_NAME)

self.sheet = self.wb.sheets[self.SHEET\_NAME]

inputs = self.sheet.range(self.RANGE).value

self.index = inputs[0].upper()

self.issue\_date = inputs[1]

self.maturity\_date = inputs[2]

self.liquidation\_date = inputs[3]

self.issue\_rate = inputs[4]

self.deal\_rate = inputs[5]

self.issuance\_par\_value = inputs[6]

self.last\_coupon = inputs[7]

self.coupon\_frequency = inputs[8]

def adjust\_dates(self):

if type(self.issue\_date) == str:

self.issue\_date = datetime.strptime(self.issue\_date, "%d/%m/%Y")

if type(self.maturity\_date) == str:

self.maturity\_date = datetime.strptime(self.maturity\_date, "%d/%m/%Y")

if type(self.liquidation\_date) == str:

self.liquidation\_date = datetime.strptime(self.liquidation\_date, "%d/%m/%Y")

if self.last\_coupon != "" and (type(self.last\_coupon) == str):

self.last\_coupon = datetime.strptime(self.last\_coupon, "%d/%m/%Y")

def check\_liquidation(self):

if self.liquidation\_date < datetime.combine(datetime.today(), time.min):

raise Personal\_exception("Liquidação passada não suportada")

class Counter\_days():

#HOLIDAY\_PATH = (r"T:\Mesa Operacoes\Credito\Narizinho\Codigos" +

# r"\Feriados anbima\holidays\_b3.feather")

HOLIDAY\_PATH = (r"C:\Victor\codes\holidays\_b3.feather")

def \_\_init\_\_(self):

self.get\_holidays()

def get\_holidays(self):

holidays\_b3 = pl.read\_ipc(self.HOLIDAY\_PATH)

holidays\_b3 = holidays\_b3.select(pl.col("Data").cast(pl.Date))

self.holidays\_numpy = holidays\_b3[:, 0].to\_numpy()

def count\_days(self, start, end, business\_days=True):

start\_string = self.test\_date(start)

end\_string = self.test\_date(end)

if business\_days:

valid\_days = "Mon Tue Wed Thu Fri"

return busday\_count(

start\_string, end\_string, holidays=self.holidays\_numpy,

weekmask=valid\_days

)

else:

valid\_days = "Mon Tue Wed Thu Fri Sat Sun"

return busday\_count(start\_string, end\_string)

def test\_date(self, date):

if type(date) == datetime:

return date.strftime("%Y-%m-%d")

elif type(date) == str:

if date[4] != "-" or int(date[5:7]) < 0 or int(date[5:7]) > 12:

raise Personal\_exception("Data em formato errado, deve ser: "

+ "aaaa-mm-dd")

return date

else:

raise Personal\_exception(

"Tentou obter dias úteis sem ser por string ou datetime")

class Yield\_curve():

'''Notation in brazil for yield is in business days'''

DICT\_MONTH = {

"F": "01", "G": "02", "H": "03", "J": "04", "K": "05", "M": "06",

"N": "07", "Q": "08", "U": "09", "V": "10", "X": "11", "Z": "12"}

URL = "https://www2.bmf.com.br/pages/portal/bmfbovespa/boletim1/Ajustes1.asp"

def \_\_init\_\_(self):

self.get\_info\_b3()

self.adjust\_text("DI de 1 dia", "Dólar comercial")

self.create\_df()

self.get\_business\_days()

self.rates\_di()

def get\_info\_b3(self):

r = requests.get(self.URL)

if r.status\_code != 200:

raise Personal\_exception(

"Não foi possível obter os dados do site da B3")

self.text = r.text

def adjust\_text(self, start\_text, end\_text):

position\_start = self.text.find(start\_text)

self.cut\_text = self.text[position\_start:]

position\_end = self.cut\_text.find(end\_text)

self.cut\_text = self.cut\_text[:position\_end]

def create\_df(self):

new\_separator = re.sub("\<.\*?\>|\r\n", ";", self.cut\_text)

splitted\_text = new\_separator.split(";")

dict\_df = {"tickers": splitted\_text[3::23],

"settle": splitted\_text[9::23]}

self.df = pl.DataFrame(dict\_df)

self.format\_df()

def format\_df(self):

self.df = self.df.with\_columns(

pl.col("tickers").str.strip(),

pl.col("settle").str.replace("\.", "")

.str.replace(",", ".").cast(pl.Float32)

)

self.df = self.df.with\_columns(

(pl.col("tickers").str.slice(0, 1).map\_dict(self.DICT\_MONTH)

.alias("maturity")

+ "-"

+ pl.col("tickers").str.slice(1, 2)

).str.strptime(pl.Date, "%m-%y")

)

def get\_business\_days(self):

self.days = Counter\_days().count\_days

self.df = self.df.with\_columns(

pl.col("maturity").dt.strftime("%Y-%m-%d").alias("business\_days")

)

self.df = self.df.with\_columns(

pl.col("business\_days")

.apply(lambda x: self.days(datetime.today(), x) + 1)

.alias("business\_days")

)

def rates\_di(self):

self.df = self.df.with\_columns(

((1e5/pl.col("settle"))\*\*(252/pl.col("business\_days")))

.alias("spot\_rate")

)

self.df = self.df.with\_columns(

(((pl.col("spot\_rate"))\*\*(

pl.col("business\_days")/252

)

/

(pl.col("spot\_rate").shift(1))\*\*(

pl.col("business\_days").shift(1)/252

)

)\*\*(252/(

pl.col("business\_days")

-

pl.col("business\_days").shift(1))

)

)

.alias("forward\_rate")

)

self.df[0,"forward\_rate"]= self.df[0, "spot\_rate"]

def create\_forward\_curve(self):

x\_di = self.df.select(pl.col("business\_days"))[:, 0].to\_numpy()

y\_di = self.df.select(pl.col("forward\_rate"))[:, 0].to\_numpy()

self.di\_curve = CubicSpline(x\_di, y\_di)

def generic\_date\_di(self, date=None, business\_days=None):

if date is not None:

self.business\_days = Counter\_days().count\_days(

datetime.today(), date)

if business\_days is not None:

self.business\_days = business\_days

else:

raise Personal\_exception(

"Incluir uma data(YYYY-MM-DD), datetime ou a quantidade de dias úteis")

self.forward\_rate = self.di\_curve(self.business\_days)

self.spot\_rate\_generic\_day3()

print(f"spot = {self.spot\_rate}, forward = {self.forward\_rate}")

return self.spot\_rate

def spot\_rate\_generic\_day(self):

print("entrou")

self.business\_days = 207

filtered\_df = self.df.filter(

pl.col("business\_days") == self.business\_days

)

if filtered\_df.is\_empty():

closest\_date = self.df.filter(

(pl.col("business\_days") - self.business\_days).abs()

==

(pl.col("business\_days") - self.business\_days)

.abs().min()

)

business\_days2 = closest\_date.select(pl.col("business\_days"))[0, 0]

spot\_rate2 = closest\_date.select(pl.col("spot\_rate"))[0, 0]

self.spot\_rate = (spot\_rate2\*\*(business\_days2/self.business\_days)

/(self.forward\_rate\*\*(

(business\_days2 - self.business\_days)/self.business\_days

))

)

else:

self.spot\_rate = filtered\_df.select(pl.col("spot\_rate"))[0, 0]

def spot\_rate\_generic\_day2(self):

filtered\_df = self.df.filter(

pl.col("business\_days") == self.business\_days

)

print("entrou2")

if filtered\_df.is\_empty():

self.df = self.df.with\_columns(

(pl.col("business\_days") - self.business\_days)

.alias("relative\_business\_days")

)

previous\_df = self.df.filter(

(pl.col("relative\_business\_days") > 0)

).filter(

(pl.col("relative\_business\_days")

== pl.col("relative\_business\_days").min()

)

)

later\_df = self.df.filter(

(pl.col("relative\_business\_days") < 0)

).filter(

(pl.col("relative\_business\_days")

== pl.col("relative\_business\_days").max()

)

)

previous\_business\_days = previous\_df.select(pl.col("business\_days"))[0, 0]

previous\_spot\_rate = previous\_df.select(pl.col("spot\_rate"))[0, 0]

later\_business\_days = later\_df.select(pl.col("business\_days"))[0, 0]

later\_spot\_rate = later\_df.select(pl.col("spot\_rate"))[0, 0]

self.spot\_rate = (previous\_spot\_rate

\*((later\_spot\_rate/previous\_spot\_rate)

\*\*((self.business\_days - previous\_business\_days)

/(later\_business\_days - previous\_business\_days))

)

)

print(self.spot\_rate)

else:

self.spot\_rate = filtered\_df.select(pl.col("spot\_rate"))[0, 0]

print(self.spot\_rate)

def spot\_rate\_generic\_day3(self, business\_days):

filtered\_df = self.df.filter(

pl.col("business\_days") == business\_days

)

if filtered\_df.is\_empty():

self.df = self.df.with\_columns(

(pl.col("business\_days") - business\_days)

.alias("relative\_business\_days")

)

previous\_df = self.df.filter(

(pl.col("relative\_business\_days") > 0)

).filter(

(pl.col("relative\_business\_days")

== pl.col("relative\_business\_days").min()

)

)

later\_df = self.df.filter(

(pl.col("relative\_business\_days") < 0)

).filter(

(pl.col("relative\_business\_days")

== pl.col("relative\_business\_days").max()

)

)

previous\_business\_days = previous\_df.select(pl.col("business\_days"))[0, 0]

previous\_spot\_rate = previous\_df.select(pl.col("spot\_rate"))[0, 0]

later\_business\_days = later\_df.select(pl.col("business\_days"))[0, 0]

later\_spot\_rate = later\_df.select(pl.col("spot\_rate"))[0, 0]

self.spot\_rate = ((previous\_spot\_rate\*\*(previous\_business\_days/business\_days))

\*(((later\_spot\_rate\*\*(later\_business\_days/business\_days))

/(previous\_spot\_rate\*\*((previous\_business\_days/business\_days))))

\*\*((business\_days - previous\_business\_days)

/(later\_business\_days - previous\_business\_days))

)

)

print(f"bu: {business\_days}, di: {self.spot\_rate}")

else:

self.spot\_rate = filtered\_df.select(pl.col("spot\_rate"))[0, 0]

print(f"bu: {business\_days}, di: {self.spot\_rate}")

class Personal\_exception(Exception):

def \_\_init\_\_(self, error\_msg):

print(error\_msg)

class Cdi\_spread():

'''Rounding rules following b3'''

def \_\_init\_\_(self, i):

self.inputs = i

self.counter = Counter\_days().count\_days

self.adjust\_coupon\_dates()

self.obtain\_cdi\_accrued()

self.calc\_current\_par\_value()

self.calc\_coupon\_values()

self.calc\_price()

def adjust\_coupon\_dates(self):

self.next\_coupon = self.inputs.last\_coupon

if self.inputs.last\_coupon == "":

self.start\_date = self.inputs.issue\_date.strftime("%Y-%m-%d")

elif self.inputs.last\_coupon > self.inputs.liquidation\_date:

self.start\_date = self.inputs.issue\_date.strftime("%Y-%m-%d")

else:

while self.next\_coupon < self.inputs.liquidation\_date:

self.inputs.last\_coupon = self.next\_coupon

self.obtain\_next\_coupon\_date()

print(self.next\_coupon)

self.start\_date = self.inputs.last\_coupon.strftime("%Y-%m-%d")

def obtain\_next\_coupon\_date(self):

m = self.next\_coupon.month + self.inputs.coupon\_frequency

y = self.next\_coupon.year

if m > 12:

y += int(m//12)

m = m%12

m = int(m)

self.next\_coupon = self.next\_coupon.replace(month=m, year=y)

def obtain\_cdi\_accrued(self):

cdi = sgs.get(('cdi', 12), start = self.start\_date, )

cdi.loc[:,"cdi\_percent"] = (cdi.loc[:,"cdi"]/100 + 1)

self.cdi\_accrued = cdi.loc[:,"cdi\_percent"].product()

self.cdi\_accrued = round(self.cdi\_accrued, 8)

print(f"cdi acumulado: {self.cdi\_accrued}")

def calc\_current\_par\_value(self):

business\_days\_to\_liquidation = self.counter(self.start\_date, self.inputs.liquidation\_date)

print(f"b.u. to liq: {business\_days\_to\_liquidation}")

issue\_accrued = (1 + self.inputs.issue\_rate)\*\*((business\_days\_to\_liquidation+2)/252)

self.yield\_factor = issue\_accrued\*self.cdi\_accrued

#pu\_par in portuguese

self.current\_par\_value = self.inputs.issuance\_par\_value\*self.yield\_factor

self.current\_par\_value = round(self.current\_par\_value, 6)

print(f"pu par: {self.current\_par\_value}")

def calc\_coupon\_values(self):

if self.next\_coupon == "":

return

self.generate\_di\_rate = Yield\_curve().spot\_rate\_generic\_day3

#first coupon

business\_days\_to\_coupon = self.counter(self.inputs.liquidation\_date, self.next\_coupon)

print(f"b.u. coupon: {business\_days\_to\_coupon}")

di\_spot = self.generate\_di\_rate(business\_days\_to\_coupon)

print(f"di: {di\_spot}")

coupon\_price = self.current\_par\_value\*((di\_spot\*(self.inputs.issue\_rate+1))\*\*(business\_days\_to\_coupon/252))

print(f"cupom futuro: {coupon\_price}")

def calc\_price(self):

business\_days\_liquidation\_to\_maturity = self.counter(self.inputs.liquidation\_date, self.inputs.maturity\_date)

print(business\_days\_liquidation\_to\_maturity)

yield\_adjust\_to\_present = ((self.inputs.issue\_rate + 1)/(self.inputs.deal\_rate + 1))\*\*(business\_days\_liquidation\_to\_maturity/252)

self.bond\_price = self.current\_par\_value\*yield\_adjust\_to\_present

self.test = yield\_adjust\_to\_present

def calc\_coupons(self):

self.y\_c = Yield\_curve()

business\_days\_first\_coupon = self.counter(self.inputs.liquidation\_date, self.inputs.next\_coupon)

# calcular o pu par de verdade, que eh o cdi acumulado do ultimo do cupom

# ate a data de liq vezes a taxa de emissao nesse periodo

#esse calc eh para deb. Checar se eh o msm para LF

class Fixed\_rate():

def \_\_init\_\_(self):

self.calc\_remaining\_par\_value()

def calc\_remaining\_par\_value(self):

business\_days\_issue\_to\_liquidation = counter(i.issue\_date, i.liquidation\_date)

issue\_accrued = (1 + i.issue\_rate)\*\*(business\_days\_issue\_to\_liquidation/252)

self.yield\_factor = issue\_accrued

self.remaining\_par\_value = i.issuance\_par\_value\*self.yield\_factor #pu\_par in portuguese

def calc\_price(self):

business\_days\_liquidation\_to\_maturity = counter(i.liquidation\_date, i.maturity\_date)

yield\_adjust\_to\_present = ((i.issue\_rate + 1)/(i.deal\_rate + 1))\*\*(business\_days\_liquidation\_to\_maturity/252)

self.bond\_price = remaining\_par\_value\*yield\_adjust\_to\_present

self.test = yield\_adjust\_to\_present

m = Main()

#bsa315 - di +, 0 amort

#alga28 - di +, 1 amort

#almc12 - di +, 3 amort

#alsca0 - di%, 0 amort