주요 모듈 소개

3. Account 데이터 추출

- Account의 결과물인 df와 jnl의 타입은 모두 pandas의 DataFrame이므로, DataFrame의 추출 방식을 동일하게 사용 가능함.
- df의 column명은 각각 df의 메서드로서 개별 column의 데이터를 추출하는 기능을 함.

In [26]:
```
from cafle import Index, Account
```

In [27]:
```
idx = Index('2023.01', 6)

cst = Account(idx)
cst.amt = 28_000
cst.addscd(idx[1], 10_000)
cst.addscd(idx[2], 10_000)
cst.addscd(idx[3],  8_000)
cst.addamt(idx[1], 8_000)
cst.addamt(idx[2], 7_000)
cst.addamt(idx[3], 7_000)
cst.addamt(idx[4], 6_000)
```

In [28]:
```
cst.df
```

Out[28]:

	bal_strt	amt_in	amt_out	bal_end
2023-01-31	0.0	0.0	0.0	0.0
2023-02-28	0.0	8000.0	0.0	8000.0
2023-03-31	8000.0	7000.0	0.0	15000.0
2023-04-30	15000.0	7000.0	0.0	22000.0
2023-05-31	22000.0	6000.0	0.0	28000.0
2023-06-30	28000.0	0.0	0.0	28000.0

pandas DataFrame의 메서드를 이용한 데이터 추출

In [29]:
```
# 'amt_in' column 추출
cst.df['amt_in']
```

Out[29]:
```
2023-01-31       0.0
2023-02-28    8000.0
2023-03-31    7000.0
2023-04-30    7000.0
2023-05-31    6000.0
2023-06-30       0.0
Name: amt_in, dtype: float64
```

In [30]:
```
# 'bal_end' column 추출
cst.df['bal_end']
```

Out[30]:
```
2023-01-31        0.0
2023-02-28     8000.0
2023-03-31    15000.0
2023-04-30    22000.0
2023-05-31    28000.0
2023-06-30    28000.0
Name: bal_end, dtype: float64
```

In [31]:
```
# 'bal_end' column의 '2023-02-28'에 해당하는 값 추출
cst.df.loc[idx[1], 'bal_end']
```

Out[31]:
```
8000.0
```

In [32]:
```
# 'bal_strt' column의 마지막 행에 해당하는 값 추출
cst.df.loc[idx[-1], 'bal_strt']
```

Out[32]:
```
28000.0
```

In [33]:
```
# 마지막 행에 해당하는 전체 값 추출
cst.df.loc[idx[-1], :]
```

Out[33]:
```
bal_strt    28000.0
amt_in          0.0
amt_out         0.0
bal_end     28000.0
Name: 2023-06-30, dtype: float64
```

In [34]:
```
from datetime import date as D
```

In [35]:
```
cst.df.loc[D(2023, 5, 31), 'amt_in']
```

Out[35]:
```
6000.0
```

In [36]:
```
cst.df.loc[idx[4], 'amt_in']
```

Out[36]:
```
6000.0
```

In [37]:
```
#슬라이싱을 이용한 추출
cst.df.loc[idx[1]:idx[4], 'bal_strt']
```

Out[37]:
```
2023-02-28        0.0
2023-03-31     8000.0
2023-04-30    15000.0
2023-05-31    22000.0
Name: bal_strt, dtype: float64
```

In [38]:
```
cst.df.loc[D(2023, 2, 28):D(2023, 5, 31), 'bal_strt']
```

Out[38]:
```
2023-02-28        0.0
2023-03-31     8000.0
2023-04-30    15000.0
2023-05-31    22000.0
Name: bal_strt, dtype: float64
```

In [ ]:
```

```

In [39]:
```
cst.jnl
```

Out[39]:

	amt_in	amt_out	rcvfrm	payto	note
2023-02-28	8000	0	None	None	add_amt
2023-03-31	7000	0	None	None	add_amt
2023-04-30	7000	0	None	None	add_amt
2023-05-31	6000	0	None	None	add_amt

In [40]:
```
cst.jnl['amt_in']
```

Out[40]:
```
2023-02-28    8000
2023-03-31    7000
2023-04-30    7000
2023-05-31    6000
Name: amt_in, dtype: object
```

In [41]:
```
cst.jnl.loc[idx[1], 'amt_in']
```

Out[41]:
```
8000
```

In [ ]:
```

```

cafle Account의 df 컬럼명은 해당 컬럼 데이터를 추출하는 메서드 역할을 함

In [42]:
```
cst.df.columns
```

Out[42]:
```
Index(['bal_strt', 'amt_in', 'amt_out', 'bal_end'], dtype='object'
)
```

In [43]:
```
cst.bal_strt[idx[0]]
```

Out[43]:
```
0.0
```

In [44]:
```
cst.amt_in[idx[1]]
```

Out[44]:
```
8000.0
```

In [45]:
```
cst.bal_end[idx[1]:idx[4]]
```

Out[45]:
```
2023-02-28     8000.0
2023-03-31    15000.0
2023-04-30    22000.0
2023-05-31    28000.0
Name: bal_end, dtype: float64
```

In [46]:
```
cst.bal_end[idx[0]:idx[-1]]
```

Out[46]:
```
2023-01-31        0.0
2023-02-28     8000.0
2023-03-31    15000.0
2023-04-30    22000.0
2023-05-31    28000.0
2023-06-30    28000.0
Name: bal_end, dtype: float64
```

In [ ]:
```

```

In [47]:
```
cst.dfall.columns
```

Out[47]:
```
Index(['scd_in', 'scd_in_cum', 'scd_out', 'scd_out_cum', 'bal_strt
', 'amt_in', 'amt_in_cum', 'amt_out', 'amt_out_cum', 'bal_end', 'r
sdl_in_cum', 'rsdl_out_cum'], dtype='object')
```

In [48]:
```
cst.scd_in[idx[3]]
```

Out[48]:
```
8000.0
```

In [49]:
```
cst.scd_in[idx[0]:idx[-1]]
```

Out[49]:
```
2023-01-31        0.0
2023-02-28    10000.0
2023-03-31    10000.0
2023-04-30     8000.0
2023-05-31        0.0
2023-06-30        0.0
Name: scd_in, dtype: float64
```

In [50]:
```
cst.rsdl_in_cum[idx[0]:idx[-1]]
```

Out[50]:
```
2023-01-31       0.0
2023-02-28    2000.0
2023-03-31    5000.0
2023-04-30    6000.0
2023-05-31       0.0
2023-06-30       0.0
Name: rsdl_in_cum, dtype: float64
```

In [ ]:
```

```

In [51]:
```
cst.jnl
```

Out[51]:

	amt_in	amt_out	rcvfrm	payto	note
2023-02-28	8000	0	None	None	add_amt
2023-03-31	7000	0	None	None	add_amt
2023-04-30	7000	0	None	None	add_amt
2023-05-31	6000	0	None	None	add_amt

In [52]:
```
cst.amt_in[idx[1]]
```

Out[52]:
```
8000.0
```

In [53]:
```
cst.amt_in[idx[1]:idx[4]]
```

Out[53]:
```
2023-02-28    8000.0
2023-03-31    7000.0
2023-04-30    7000.0
2023-05-31    6000.0
Name: amt_in, dtype: float64
```

In [ ]:
```

```

4. Account 다중 설정

- Account를 다중 설정하여 객체의 대분류, 소분류 구분이 가능함
- Account.**subacc**(*subacc명*) : Account에 하위 Account를 설정하고, 이를 _subacc_라고 명칭함
- Account.**dct** : 하위 Account의 dictionary
- Account.**mrg** : 하위 Account의 현금흐름을 취합

공사비를 대분류로, 토목공사비/건축공사비를 소분류로 구분

In [54]:
```
공사비 = Account(idx)
공사비.토목공사비 = 공사비.subacc("토목공사비")
공사비.건축공사비 = 공사비.subacc("건축공사비")
```

In [55]:
```
공사비
```

Out[55]:
```
Account(main, len 6, dct: ['토목공사비', '건축공사비'])
```

In [56]:
```
공사비.토목공사비
```

Out[56]:
```
Account(토목공사비, len 6)
```

In [57]:
```
공사비.건축공사비
```

Out[57]:
```
Account(건축공사비, len 6)
```

In [58]:
```
공사비.dct
```

Out[58]:
```
{'토목공사비': Account(토목공사비, len 6), '건축공사비': Account(건축공사비,
len 6)}
```

In [59]:
```
공사비.dct['토목공사비']
```

Out[59]:
```
Account(토목공사비, len 6)
```

In [60]:
```
공사비.dct['건축공사비']
```

Out[60]:
```
Account(건축공사비, len 6)
```

In [ ]:
```

```

In [61]:
```
with 공사비.토목공사비 as c:
    c.addamt(idx, [100, 100, 100, 100, 100, 100])
```

In [62]:
```
공사비.토목공사비.df
```

Out[62]:

	bal_strt	amt_in	amt_out	bal_end
2023-01-31	0.0	100.0	0.0	100.0
2023-02-28	100.0	100.0	0.0	200.0
2023-03-31	200.0	100.0	0.0	300.0
2023-04-30	300.0	100.0	0.0	400.0
2023-05-31	400.0	100.0	0.0	500.0
2023-06-30	500.0	100.0	0.0	600.0

In [63]:
```
공사비.토목공사비.jnl
```

Out[63]:

	amt_in	amt_out	rcvfrm	payto	note
2023-01-31	100	0	None	None	add_amt
2023-02-28	100	0	None	None	add_amt
2023-03-31	100	0	None	None	add_amt
2023-04-30	100	0	None	None	add_amt
2023-05-31	100	0	None	None	add_amt
2023-06-30	100	0	None	None	add_amt

In [ ]:
```

```

In [64]:
```
with 공사비.건축공사비 as c:
    c.addamt(idx, [300, 300, 300, 300, 300, 300])
```

In [65]:
```
공사비.건축공사비.df
```

Out[65]:

	bal_strt	amt_in	amt_out	bal_end
2023-01-31	0.0	300.0	0.0	300.0
2023-02-28	300.0	300.0	0.0	600.0
2023-03-31	600.0	300.0	0.0	900.0
2023-04-30	900.0	300.0	0.0	1200.0
2023-05-31	1200.0	300.0	0.0	1500.0
2023-06-30	1500.0	300.0	0.0	1800.0

In [66]:
```
공사비.건축공사비.jnl
```

Out[66]:

	amt_in	amt_out	rcvfrm	payto	note
2023-01-31	300	0	None	None	add_amt
2023-02-28	300	0	None	None	add_amt
2023-03-31	300	0	None	None	add_amt
2023-04-30	300	0	None	None	add_amt
2023-05-31	300	0	None	None	add_amt
2023-06-30	300	0	None	None	add_amt

In [ ]:
```
```

In [67]:
```
공사비.mrg
```

Out[67]:
```
Account(main, len 6)
```

In [68]:
```
공사비.mrg.df
```

Out[68]:

	bal_strt	amt_in	amt_out	bal_end
2023-01-31	0.0	400.0	0.0	400.0
2023-02-28	400.0	400.0	0.0	800.0
2023-03-31	800.0	400.0	0.0	1200.0
2023-04-30	1200.0	400.0	0.0	1600.0
2023-05-31	1600.0	400.0	0.0	2000.0
2023-06-30	2000.0	400.0	0.0	2400.0

In [69]:
```
공사비.mrg.jnl
```

Out[69]:

	amt_in	amt_out	rcvfrm	payto	note
2023-01-31	100	0	None	None	add_amt
2023-01-31	300	0	None	None	add_amt
2023-02-28	100	0	None	None	add_amt
2023-02-28	300	0	None	None	add_amt
2023-03-31	100	0	None	None	add_amt
2023-03-31	300	0	None	None	add_amt
2023-04-30	100	0	None	None	add_amt
2023-04-30	300	0	None	None	add_amt
2023-05-31	100	0	None	None	add_amt
2023-05-31	300	0	None	None	add_amt
2023-06-30	100	0	None	None	add_amt
2023-06-30	300	0	None	None	add_amt

In [ ]:
```
```

보통주를 대분류로, 주주1과 주주2를 소분류로 구분

In [70]:
```
idx = Index("2023.01", 6)
```

In [71]:
```
보통주 = Account(idx)
```

In [72]:
```
보통주.주주1 = 보통주.subacc("주주1")
with 보통주.주주1 as e:
    e.amt = 5_000
    e.subscd(idx[0], e.amt, note="주주1의 최초 출자금")
    e.addscd(idx[-1], e.amt, note="주주1의 출자금 회수")
```

In [73]:
```
보통주.주주2 = 보통주.subacc("주주2")
with 보통주.주주2 as e:
    e.amt = 3_000
    e.subscd(idx[0], e.amt, note="주주2의 최초 출자금")
    e.addscd(idx[-1], e.amt, note="주주2의 출자금 회수")
```

In [74]:
```
보통주
```

Out[74]:
```
Account(main, len 6, dct: ['주주1', '주주2'])
```

In [75]:
```
보통주.주주1.dfall
```

Out[75]:

	scd_in	scd_in_cum	scd_out	scd_out_cum	bal_strt	amt_in	amt_in_cum	amt_out
2023-01-31	0.0	0.0	5000.0	5000.0	0.0	0.0	0.0	0.0
2023-02-28	0.0	0.0	0.0	5000.0	0.0	0.0	0.0	0.0
2023-03-31	0.0	0.0	0.0	5000.0	0.0	0.0	0.0	0.0
2023-04-30	0.0	0.0	0.0	5000.0	0.0	0.0	0.0	0.0
2023-05-31	0.0	0.0	0.0	5000.0	0.0	0.0	0.0	0.0
2023-06-30	5000.0	5000.0	0.0	5000.0	0.0	0.0	0.0	0.0

In [76]:
```
보통주.주주2.dfall
```

Out[76]:

	scd_in	scd_in_cum	scd_out	scd_out_cum	bal_strt	amt_in	amt_in_cum	amt_out
2023-01-31	0.0	0.0	3000.0	3000.0	0.0	0.0	0.0	0.0
2023-02-28	0.0	0.0	0.0	3000.0	0.0	0.0	0.0	0.0
2023-03-31	0.0	0.0	0.0	3000.0	0.0	0.0	0.0	0.0
2023-04-30	0.0	0.0	0.0	3000.0	0.0	0.0	0.0	0.0
2023-05-31	0.0	0.0	0.0	3000.0	0.0	0.0	0.0	0.0
2023-06-30	3000.0	3000.0	0.0	3000.0	0.0	0.0	0.0	0.0

In [ ]:
```

```

In [77]:
```
oprtg = Account(idx) #운영계좌
```

In [ ]:
```

```

In [78]:
```
#idx[0]
보통주.주주1.send(idx[0], 보통주.주주1.scd_out[idx[0]], oprtg, note="주주1 출자금")
보통주.주주2.send(idx[0], 보통주.주주2.scd_out[idx[0]], oprtg, note="주주2 출자금")
```

In [79]:
```
#idx[5]
oprtg.send(idx[5], 보통주.주주1.scd_in[idx[5]], 보통주.주주1, note="주주1 분배")
oprtg.send(idx[5], 보통주.주주2.scd_in[idx[5]], 보통주.주주2, note="주주2 분배")
```

In [80]:
```
oprtg.df
```

Out[80]:

	bal_strt	amt_in	amt_out	bal_end
2023-01-31	0.0	8000.0	0.0	8000.0
2023-02-28	8000.0	0.0	0.0	8000.0
2023-03-31	8000.0	0.0	0.0	8000.0
2023-04-30	8000.0	0.0	0.0	8000.0
2023-05-31	8000.0	0.0	0.0	8000.0
2023-06-30	8000.0	0.0	8000.0	0.0

In [81]:
```
oprtg.jnl
```

Out[81]:

	amt_in	amt_out	rcvfrm	payto	note
2023-01-31	5000.0	0	주주1	None	주주1 출자금
2023-01-31	3000.0	0	주주2	None	주주2 출자금
2023-06-30	0.0	5000.0	None	주주1	주주1 분배
2023-06-30	0.0	3000.0	None	주주2	주주2 분배

In [82]:
```
보통주.주주1.dfall
```

Out[82]:

	scd_in	scd_in_cum	scd_out	scd_out_cum	bal_strt	amt_in	amt_in_cum	amt_out
2023-01-31	0.0	0.0	5000.0	5000.0	0.0	0.0	0.0	5000.0
2023-02-28	0.0	0.0	0.0	5000.0	-5000.0	0.0	0.0	0.0
2023-03-31	0.0	0.0	0.0	5000.0	-5000.0	0.0	0.0	0.0
2023-04-30	0.0	0.0	0.0	5000.0	-5000.0	0.0	0.0	0.0
2023-05-31	0.0	0.0	0.0	5000.0	-5000.0	0.0	0.0	0.0
2023-06-30	5000.0	5000.0	0.0	5000.0	-5000.0	5000.0	5000.0	0.0

In [83]:
```
보통주.주주1.jnlscd
```

Out[83]:

	amt_in	amt_out	rcvfrm	payto	note
2023-01-31	0	5000	None	None	주주1의 최초 출자금
2023-06-30	5000	0	None	None	주주1의 출자금 회수

In [84]:
```
보통주.주주1.jnl
```

Out[84]:

	amt_in	amt_out	rcvfrm	payto	note
2023-01-31	0	5000.0	None	main	주주1 출자금
2023-06-30	5000.0	0.0	main	None	주주1 분배

In [85]:
```
보통주.주주2.dfall
```

Out[85]:

	scd_in	scd_in_cum	scd_out	scd_out_cum	bal_strt	amt_in	amt_in_cum	amt_out
2023-01-31	0.0	0.0	3000.0	3000.0	0.0	0.0	0.0	3000.0
2023-02-28	0.0	0.0	0.0	3000.0	-3000.0	0.0	0.0	0.0
2023-03-31	0.0	0.0	0.0	3000.0	-3000.0	0.0	0.0	0.0
2023-04-30	0.0	0.0	0.0	3000.0	-3000.0	0.0	0.0	0.0
2023-05-31	0.0	0.0	0.0	3000.0	-3000.0	0.0	0.0	0.0
2023-06-30	3000.0	3000.0	0.0	3000.0	-3000.0	3000.0	3000.0	0.0

In [86]:
```
보통주.주주2.jnlscd
```

Out[86]:

	amt_in	amt_out	rcvfrm	payto	note
2023-01-31	0	3000	None	None	주주2의 최초 출자금
2023-06-30	3000	0	None	None	주주2의 출자금 회수

In [87]:
```
보통주.주주2.jnl
```

Out[87]:

	amt_in	amt_out	rcvfrm	payto	note
2023-01-31	0	3000.0	None	main	주주2 출자금
2023-06-30	3000.0	0.0	main	None	주주2 분배

In [ ]:
```

```

for문을 이용한 현금흐름 작성

In [88]:
```
보통주 = Account(idx)

보통주.주주1 = 보통주.subacc("주주1")
with 보통주.주주1 as e:
    e.amt = 5_000
    e.subscd(idx[0], e.amt, note="주주1의 최초 출자금")
    e.addscd(idx[-1], e.amt, note="주주1의 출자금 회수")

보통주.주주2 = 보통주.subacc("주주2")
with 보통주.주주2 as e:
    e.amt = 3_000
    e.subscd(idx[0], e.amt, note="주주2의 최초 출자금")
    e.addscd(idx[-1], e.amt, note="주주2의 출자금 회수")

oprtg = Account(idx) #운영계좌
```

In [89]:
```
for i in idx:
    보통주.주주1.send(i, 보통주.주주1.scd_out[i], oprtg, note="주주1 출자금"
)
    보통주.주주2.send(i, 보통주.주주2.scd_out[i], oprtg, note="주주2 출자금"
)
    oprtg.send(i, 보통주.주주1.scd_in[i], 보통주.주주1, note="주주1 분배")
    oprtg.send(i, 보통주.주주2.scd_in[i], 보통주.주주2, note="주주2 분배")
```

In [90]:
```
oprtg.jnl
```

Out[90]:

	amt_in	amt_out	rcvfrm	payto	note
2023-01-31	5000.0	0	주주1	None	주주1 출자금
2023-01-31	3000.0	0	주주2	None	주주2 출자금
2023-06-30	0.0	5000.0	None	주주1	주주1 분배
2023-06-30	0.0	3000.0	None	주주2	주주2 분배

In [91]:
```
oprtg.df
```

Out[91]:

	bal_strt	amt_in	amt_out	bal_end
2023-01-31	0.0	8000.0	0.0	8000.0
2023-02-28	8000.0	0.0	0.0	8000.0
2023-03-31	8000.0	0.0	0.0	8000.0
2023-04-30	8000.0	0.0	0.0	8000.0
2023-05-31	8000.0	0.0	0.0	8000.0
2023-06-30	8000.0	0.0	8000.0	0.0

In [ ]:

금융조달 측면에서 다중 Account 설정 이용

- 재무모델 작성시 금융조달 측면에서 여러가지 현금흐름이 동시에 필요함.
  - 대출원금 : 최초 대출금의 인출, 대출기간 중에 발생하는 한도인출, 대출기간 중 발생되는 중도상환, 만기시 발생되는 만기상환 등을 처리
  - 이자 : 대출기간 중 인출된 대출금에 대한 사용 비용을 계산하고, 해당 이자금원의 유출입을 처리
  - 수수료 : 대출 실행시 일반적으로 대출을 실행하고, 인출하는 데 대한 수수료를 청구하며, 이에 대한 유출입을 처리
  - 미인출수수료 : 한도대출이 실행되는 경우 미인출된 대출금 한도에 대해서는 자금의 대기비용을 계산하여 별도의 미인출수수료를 청구하기도 함.
- 하나의 금융조달 객체(보통주, 우선주, 대출 등)에 대하여 상기와 같이 필요한 개별 Account들을 하위 Account로 설정하여 사용하면 데이터의 관리 및 추출에 편리함.

In [92]:
```
idx = Index("2023.01", 6)
```

In [93]:
```
loan = Account(idx)
loan.mtrt = len(idx) - 1 #만기 설정
```

In [94]:
```
# 대출금 현금흐름 설정
loan.ntnl = loan.subacc("ntnl")
with loan.ntnl as n:
    n.amt = 10_000
    n.subscd(idx[0], n.amt)
    n.addscd(idx[-1], n.amt)
```

In [95]:
```
# 대출금 취급수수료 설정
loan.fee = loan.subacc("fee")
with loan.fee as f:
    f.rate = 0.02 #2.0%
    f.amt = f.rate * loan.ntnl.amt
    f.addscd(idx[0], f.amt)
```

In [96]:
```
# 대출금 이자 설정
loan.IR = loan.subacc("IR")
with loan.IR as i:
    i.rate = 0.06 #6.0%
    i.cycle = 1 #1개월 마다 이자 지급
    i.rate_cycle = i.rate / 12 * i.cycle #1개월 기준으로 계산되는 이자율
```

In [97]:
```
loan
```

Out[97]:
```
Account(main, len 6, dct: ['ntnl', 'fee', 'IR'])
```

In [98]:
```
print("대출금 : ", loan.ntnl.amt)
print("이자율 : ", loan.IR.rate)
print("수수료율 : ", loan.fee.rate)
```

```
대출금 :  10000
이자율 :  0.06
수수료율 :  0.02
```

In [99]:
```
loan.fee.amt
```

Out[99]:
```
200.0
```

In [ ]:
```
```

In [100]:
```
oprtg = Account(idx)
oprtg.addamt(idx[0], 1000, note="최초 계좌 잔액")
```

In [101]:
```
oprtg.df
```

Out[101]:

	bal_strt	amt_in	amt_out	bal_end
2023-01-31	0.0	1000.0	0.0	1000.0
2023-02-28	1000.0	0.0	0.0	1000.0
2023-03-31	1000.0	0.0	0.0	1000.0
2023-04-30	1000.0	0.0	0.0	1000.0
2023-05-31	1000.0	0.0	0.0	1000.0
2023-06-30	1000.0	0.0	0.0	1000.0

In [ ]:
```
```

In [102]:
```
#idx[0] : 대출금의 인출 및 수수료 수취
loan.ntnl.send(idx[0], loan.ntnl.scd_out[idx[0]], oprtg, note="대출금 인출")
oprtg.send(idx[0], loan.fee.scd_in[idx[0]], loan.fee, note="취급수수료")
```

In [103]:
```
#idx[1] : 이자 계산 및 이자 수취
IR_amt = -loan.ntnl.bal_strt[idx[1]] * loan.IR.rate_cycle
oprtg.send(idx[1], IR_amt, loan.IR, note="대출이자")
```

In [104]:
```
#idx[2] : 이자 계산 및 이자 수취
IR_amt = -loan.ntnl.bal_strt[idx[2]] * loan.IR.rate_cycle
oprtg.send(idx[2], IR_amt, loan.IR, note="대출이자")
```

In [105]:
```
#idx[3] : 이자 계산 및 이자 수취
IR_amt = -loan.ntnl.bal_strt[idx[3]] * loan.IR.rate_cycle
oprtg.send(idx[3], IR_amt, loan.IR, note="대출이자")
```

In [106]:
```
#idx[4] : 이자 계산 및 이자 수취
IR_amt = -loan.ntnl.bal_strt[idx[4]] * loan.IR.rate_cycle
oprtg.send(idx[4], IR_amt, loan.IR, note="대출이자")
```

In [107]:
```
#idx[5] : 이자 계산 및 이자 수취, 대출원금 회수
IR_amt = -loan.ntnl.bal_strt[idx[5]] * loan.IR.rate_cycle
oprtg.send(idx[5], IR_amt, loan.IR, note="대출이자")
oprtg.send(idx[5], loan.ntnl.scd_in[idx[5]], loan.ntnl, note="대출금 상환")
```

In [108]:
```
oprtg.jnl
```

Out[108]:

	amt_in	amt_out	rcvfrm	payto	note
2023-01-31	1000	0	None	None	최초 계좌 잔액
2023-01-31	10000.0	0	ntnl	None	대출금 인출
2023-01-31	0	200.0	None	fee	취급수수료
2023-02-28	0	50.0	None	IR	대출이자
2023-03-31	0	50.0	None	IR	대출이자
2023-04-30	0	50.0	None	IR	대출이자
2023-05-31	0	50.0	None	IR	대출이자
2023-06-30	0	50.0	None	IR	대출이자
2023-06-30	0	10000.0	None	ntnl	대출금 상환

In [109]:
```
oprtg.df
```

Out[109]:

	bal_strt	amt_in	amt_out	bal_end
2023-01-31	0.0	11000.0	200.0	10800.0
2023-02-28	10800.0	0.0	50.0	10750.0
2023-03-31	10750.0	0.0	50.0	10700.0
2023-04-30	10700.0	0.0	50.0	10650.0
2023-05-31	10650.0	0.0	50.0	10600.0
2023-06-30	10600.0	0.0	10050.0	550.0

In [110]:
```
loan.ntnl.dfall
```

Out[110]:

	scd_in	scd_in_cum	scd_out	scd_out_cum	bal_strt	amt_in	amt_in_cum	amt_o
2023-01-31	0.0	0.0	10000.0	10000.0	0.0	0.0	0.0	10000
2023-02-28	0.0	0.0	0.0	10000.0	-10000.0	0.0	0.0	0
2023-03-31	0.0	0.0	0.0	10000.0	-10000.0	0.0	0.0	0
2023-04-30	0.0	0.0	0.0	10000.0	-10000.0	0.0	0.0	0
2023-05-31	0.0	0.0	0.0	10000.0	-10000.0	0.0	0.0	0
2023-06-30	10000.0	10000.0	0.0	10000.0	-10000.0	10000.0	10000.0	0

In [111]:
```
loan.ntnl.jnl
```

Out[111]:

	amt_in	amt_out	rcvfrm	payto	note
2023-01-31	0	10000.0	None	main	대출금 인출
2023-06-30	10000.0	0.0	main	None	대출금 상환

In [112]:
```
loan.IR.df
```

Out[112]:

	bal_strt	amt_in	amt_out	bal_end
2023-01-31	0.0	0.0	0.0	0.0
2023-02-28	0.0	50.0	0.0	50.0
2023-03-31	50.0	50.0	0.0	100.0
2023-04-30	100.0	50.0	0.0	150.0
2023-05-31	150.0	50.0	0.0	200.0
2023-06-30	200.0	50.0	0.0	250.0

In [113]:
```
loan.IR.jnl
```

Out[113]:

	amt_in	amt_out	rcvfrm	payto	note
2023-02-28	50.0	0	main	None	대출이자
2023-03-31	50.0	0	main	None	대출이자
2023-04-30	50.0	0	main	None	대출이자
2023-05-31	50.0	0	main	None	대출이자
2023-06-30	50.0	0	main	None	대출이자

In [114]:
```
loan.fee.df
```

Out[114]:

	bal_strt	amt_in	amt_out	bal_end
2023-01-31	0.0	200.0	0.0	200.0
2023-02-28	200.0	0.0	0.0	200.0
2023-03-31	200.0	0.0	0.0	200.0
2023-04-30	200.0	0.0	0.0	200.0
2023-05-31	200.0	0.0	0.0	200.0
2023-06-30	200.0	0.0	0.0	200.0

In [115]:
```
loan.fee.jnl
```

Out[115]:

	amt_in	amt_out	rcvfrm	payto	note
2023-01-31	200.0	0	main	None	취급수수료

In [116]:
```
loan.mrg.df
```

Out[116]:

	bal_strt	amt_in	amt_out	bal_end
2023-01-31	0.0	200.0	10000.0	-9800.0
2023-02-28	-9800.0	50.0	0.0	-9750.0
2023-03-31	-9750.0	50.0	0.0	-9700.0
2023-04-30	-9700.0	50.0	0.0	-9650.0
2023-05-31	-9650.0	50.0	0.0	-9600.0
2023-06-30	-9600.0	10050.0	0.0	450.0

In [ ]:
```

```

for문으로 대출 현금흐름 추정

In [117]:
```
idx = Index("2023.01", 6)

loan = Account(idx)
loan.mtrt = len(idx) - 1 #만기 설정

# 대출금 현금흐름 설정
loan.ntnl = loan.subacc("ntnl")
with loan.ntnl as n:
    n.amt = 10_000
    n.subscd(idx[0], n.amt)
    n.addscd(idx[-1], n.amt)

# 대출금 취급수수료 설정
loan.fee = loan.subacc("fee")
with loan.fee as f:
    f.rate = 0.02 #2.0%
    f.amt = f.rate * loan.ntnl.amt
    f.addscd(idx[0], f.amt)

# 대출금 이자 설정
loan.IR = loan.subacc("IR")
with loan.IR as i:
    i.rate = 0.06 #6.0%
    i.cycle = 1 #1개월 마다 이자 지급
    i.rate_cycle = i.rate / 12 * i.cycle #1개월 기준으로 계산되는 이자율

# 운영계좌 설정
oprtg = Account(idx)
oprtg.addamt(idx[0], 1000, note="최초 계좌 잔액")
```

In [118]:
```
for i in idx:
    #대출금의 인출 및 수수료 수취
    loan.ntnl.send(i, loan.ntnl.scd_out[i], oprtg, note="대출금 인출")
    oprtg.send(i, loan.fee.scd_in[i], loan.fee, note="취급수수료")

    #이자 계산 및 이자 수취
    IR_amt = -loan.ntnl.bal_strt[i] * loan.IR.rate_cycle
    oprtg.send(i, IR_amt, loan.IR, note="대출이자")

    #대출원금 회수
    oprtg.send(i, loan.ntnl.scd_in[i], loan.ntnl, note="대출금 상환")
```

In [119]:
```
oprtg.df
```

Out[119]:

	bal_strt	amt_in	amt_out	bal_end
2023-01-31	0.0	11000.0	200.0	10800.0
2023-02-28	10800.0	0.0	50.0	10750.0
2023-03-31	10750.0	0.0	50.0	10700.0
2023-04-30	10700.0	0.0	50.0	10650.0
2023-05-31	10650.0	0.0	50.0	10600.0
2023-06-30	10600.0	0.0	10050.0	550.0

In [120]:
```
oprtg.jnl
```

Out[120]:

	amt_in	amt_out	rcvfrm	payto	note
2023-01-31	1000	0	None	None	최초 계좌 잔액
2023-01-31	10000.0	0	ntnl	None	대출금 인출
2023-01-31	0	200.0	None	fee	취급수수료
2023-02-28	0	50.0	None	IR	대출이자
2023-03-31	0	50.0	None	IR	대출이자
2023-04-30	0	50.0	None	IR	대출이자
2023-05-31	0	50.0	None	IR	대출이자
2023-06-30	0	50.0	None	IR	대출이자
2023-06-30	0	10000.0	None	ntnl	대출금 상환

In [ ]:
```

```