PYCON 2018

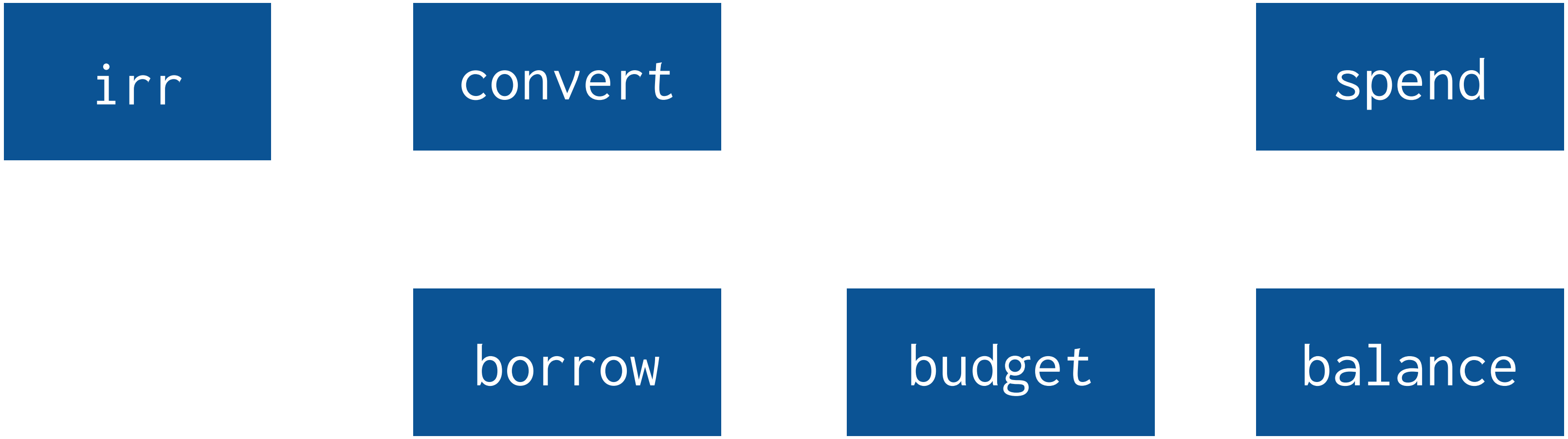
The Art of Coding

personal finance

@maxhumber

personal **py**nance

@maxhumber



irr

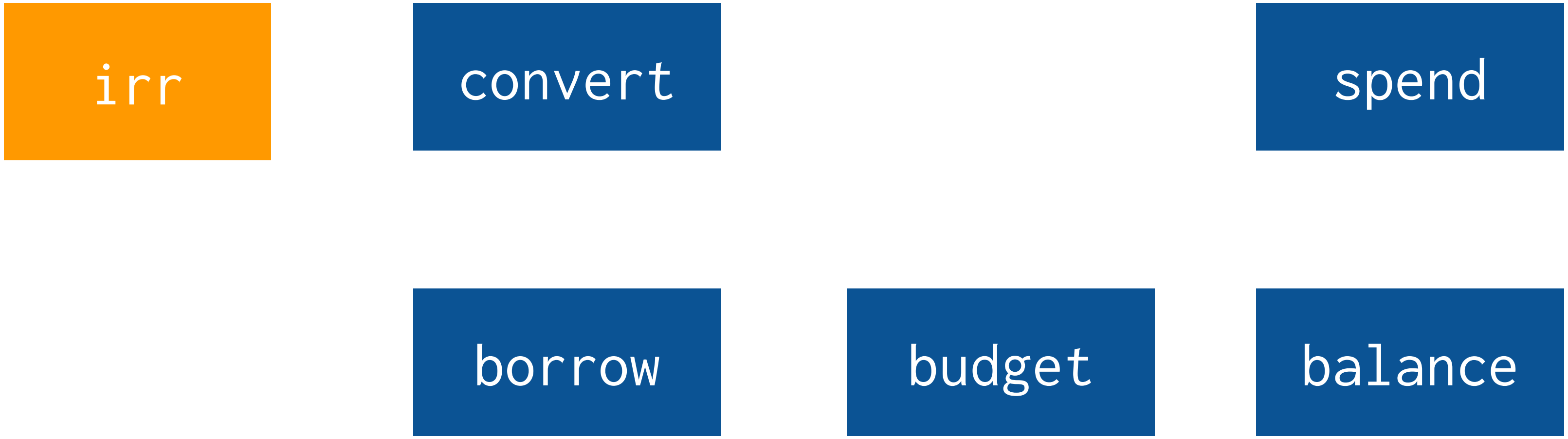
convert

spend

borrow

budget

balance



irr

convert

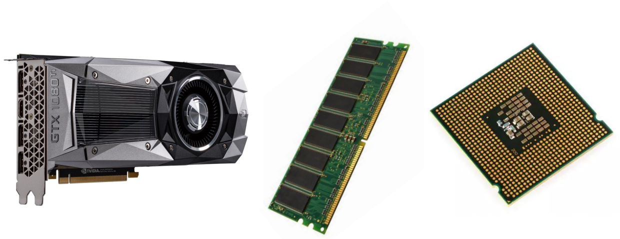
spend

borrow

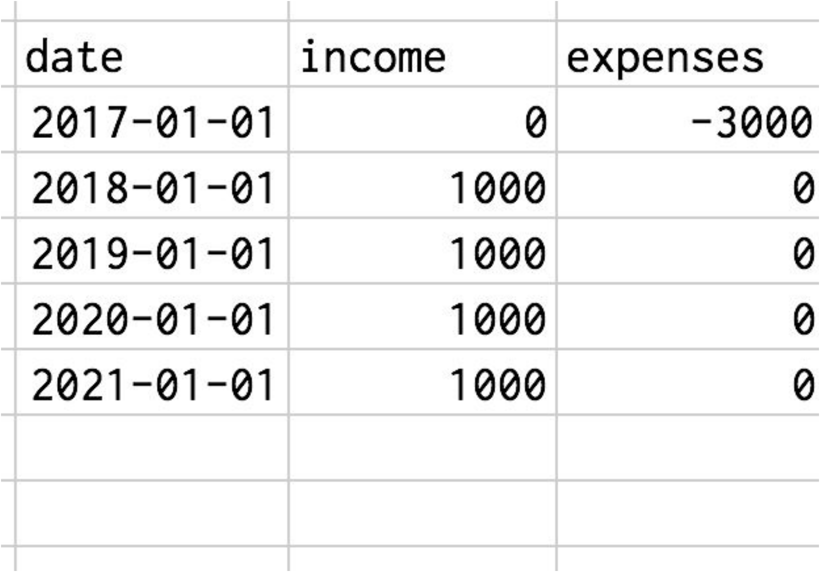
budget

balance

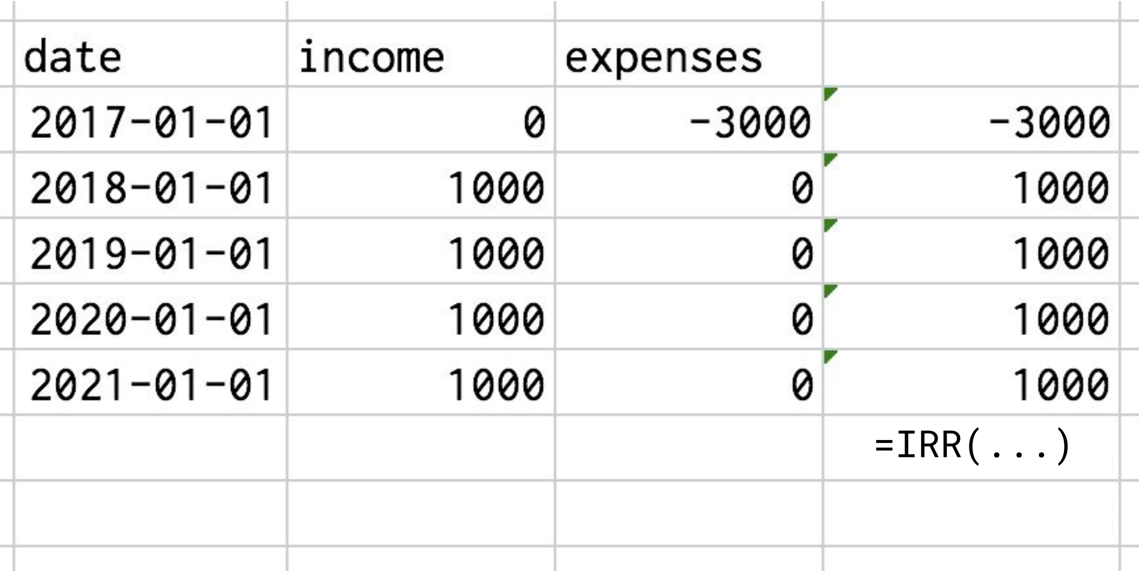
1+2=?



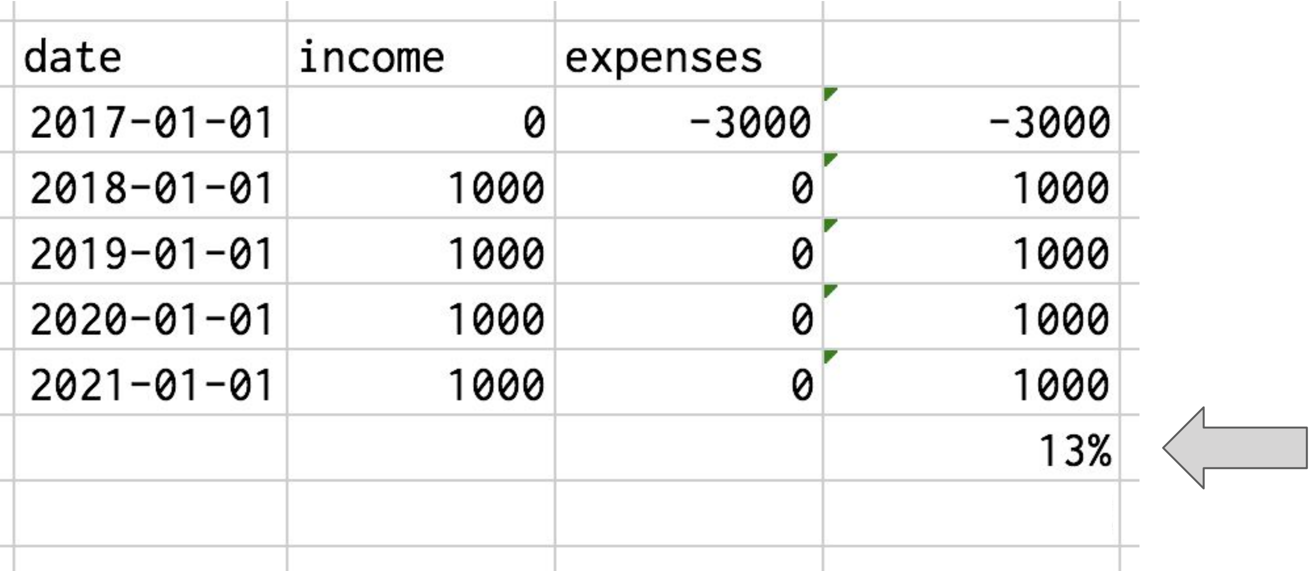
$3000



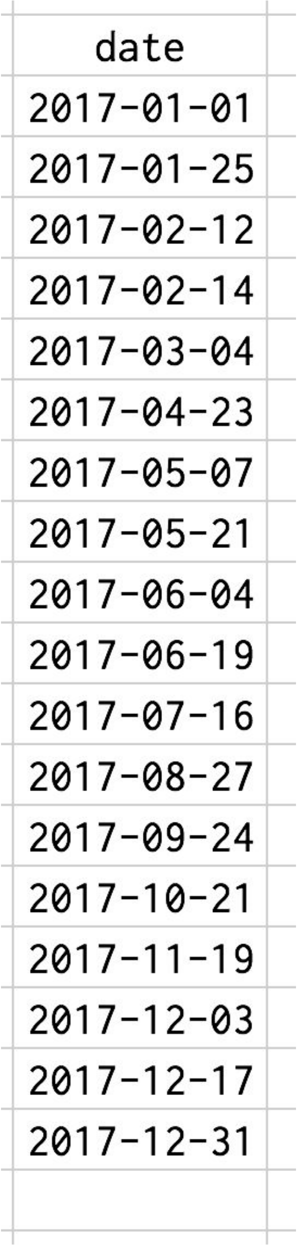
date	income	expenses
2017-01-01	0	-3000
2018-01-01	1000	0
2019-01-01	1000	0
2020-01-01	1000	0
2021-01-01	1000	0



date	income	expenses	
2017-01-01	0	-3000	-3000
2018-01-01	1000	0	1000
2019-01-01	1000	0	1000
2020-01-01	1000	0	1000
2021-01-01	1000	0	1000
			=IRR(...)

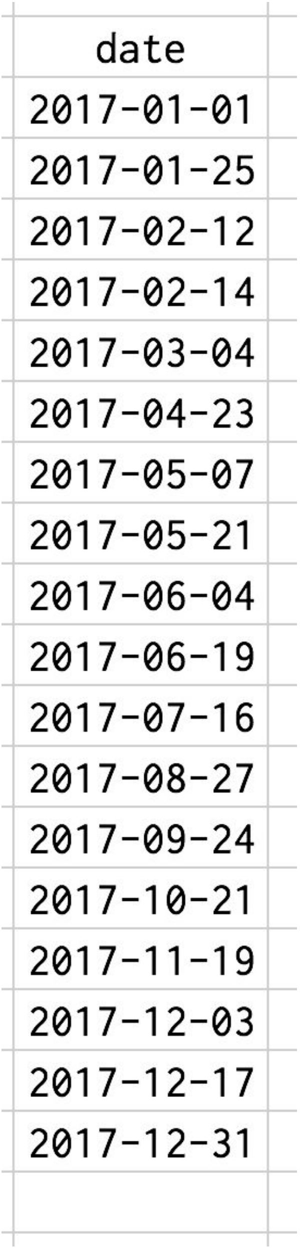


date	income	expenses	
2017-01-01	0	-3000	-3000
2018-01-01	1000	0	1000
2019-01-01	1000	0	1000
2020-01-01	1000	0	1000
2021-01-01	1000	0	1000
			13%



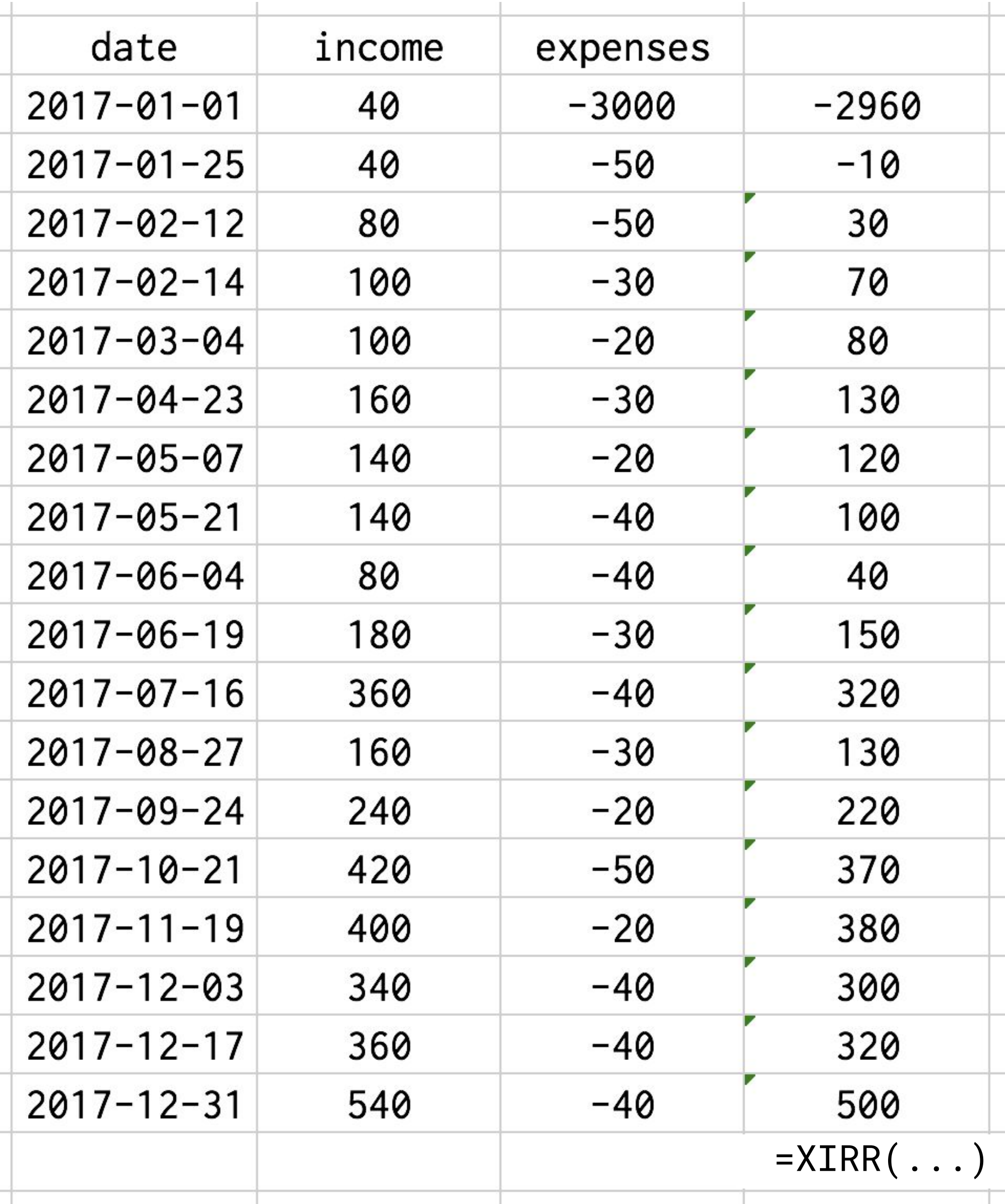
date
2017-01-01
2017-01-25
2017-02-12
2017-02-14
2017-03-04
2017-04-23
2017-05-07
2017-05-21
2017-06-04
2017-06-19
2017-07-16
2017-08-27
2017-09-24
2017-10-21
2017-11-19
2017-12-03
2017-12-17
2017-12-31

date
2017-01-01
2017-01-25
2017-02-12
2017-02-14
2017-03-04
2017-04-23
2017-05-07
2017-05-21
2017-06-04
2017-06-19
2017-07-16
2017-08-27
2017-09-24
2017-10-21
2017-11-19
2017-12-03
2017-12-17
2017-12-31

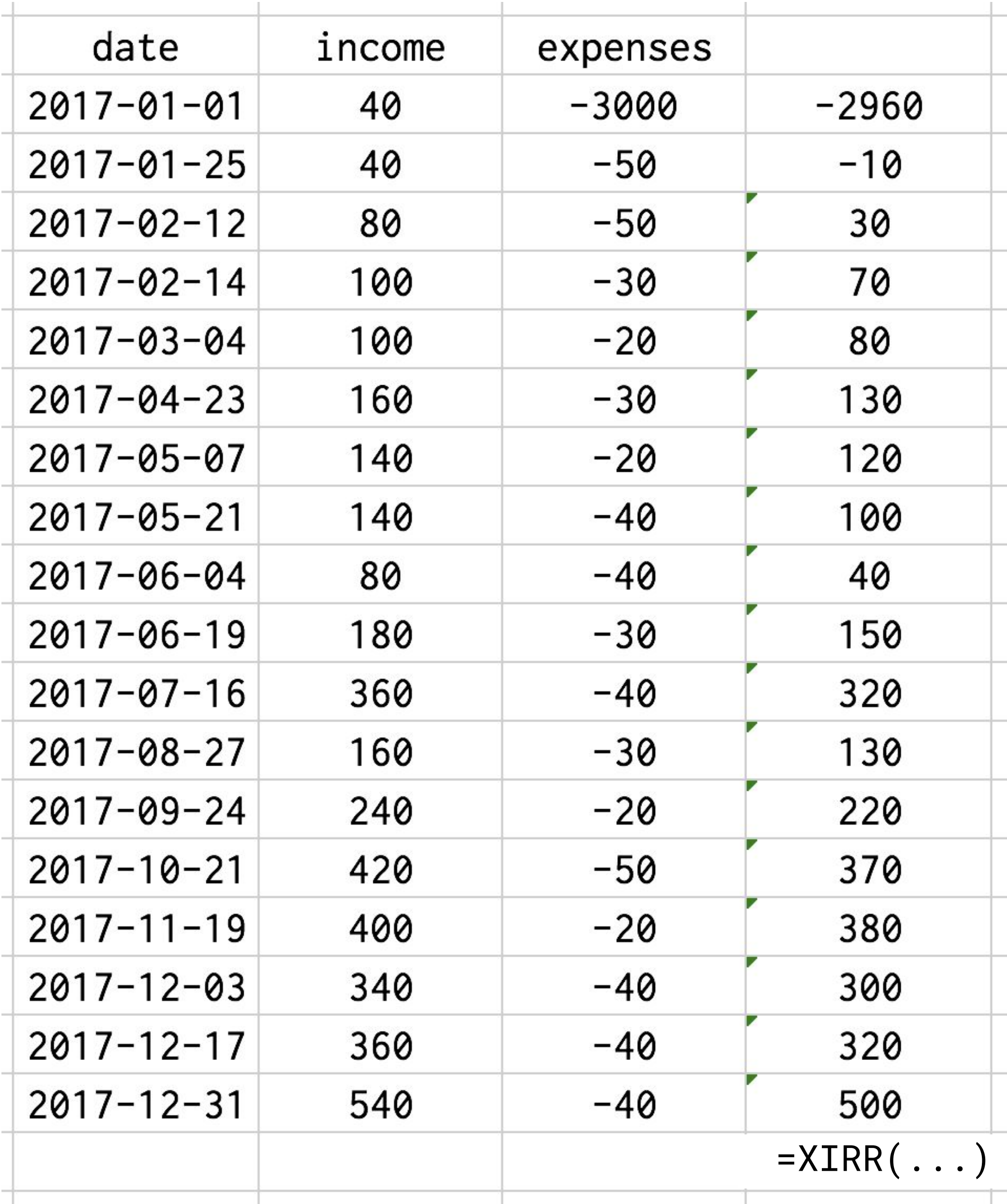


date
2017-01-01
2017-01-25
2017-02-12
2017-02-14
2017-03-04
2017-04-23
2017-05-07
2017-05-21
2017-06-04
2017-06-19
2017-07-16
2017-08-27
2017-09-24
2017-10-21
2017-11-19
2017-12-03
2017-12-17
2017-12-31

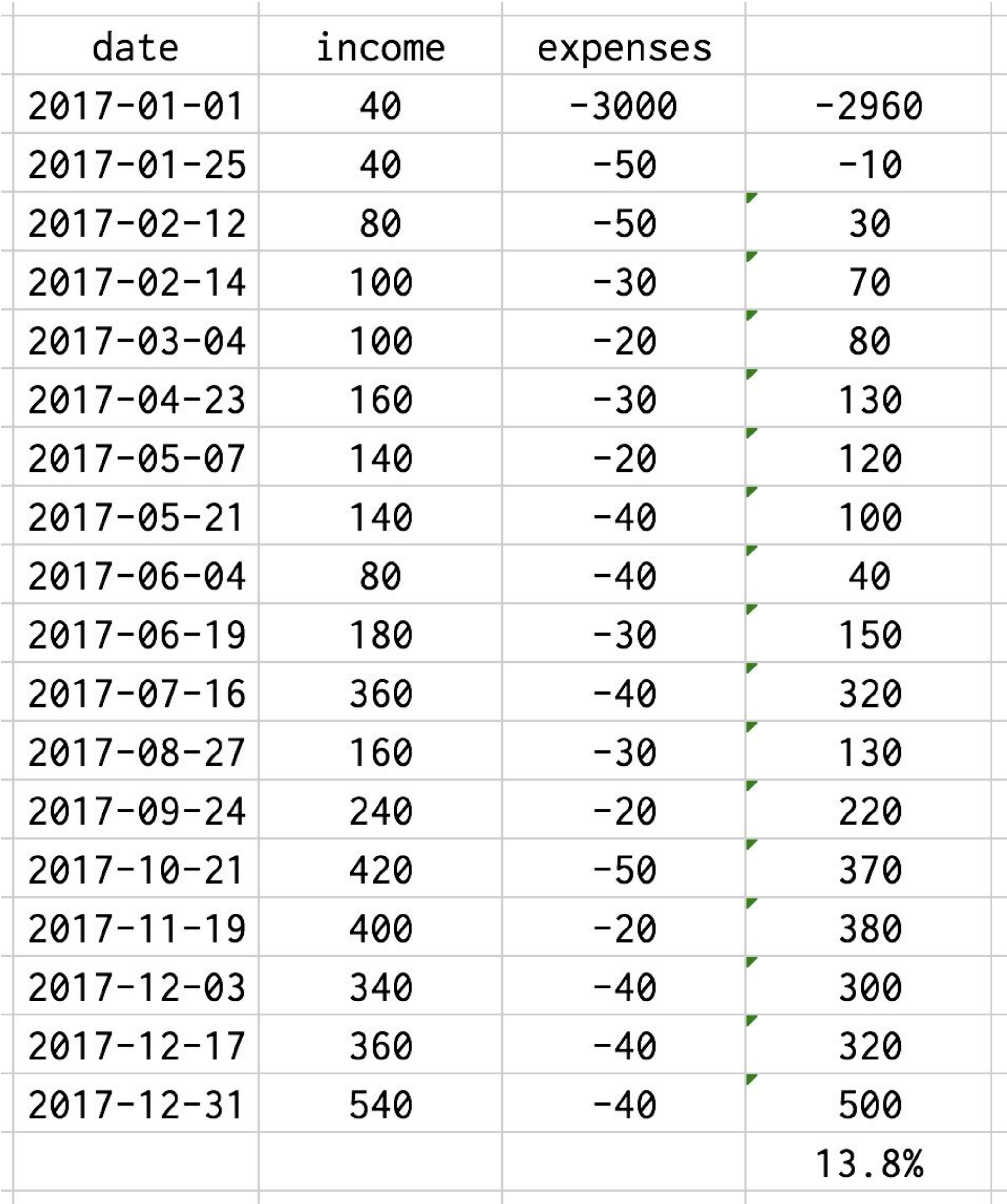
```
=XIRR([v],[d])
```



date	income	expenses	
2017-01-01	40	-3000	-2960
2017-01-25	40	-50	-10
2017-02-12	80	-50	30
2017-02-14	100	-30	70
2017-03-04	100	-20	80
2017-04-23	160	-30	130
2017-05-07	140	-20	120
2017-05-21	140	-40	100
2017-06-04	80	-40	40
2017-06-19	180	-30	150
2017-07-16	360	-40	320
2017-08-27	160	-30	130
2017-09-24	240	-20	220
2017-10-21	420	-50	370
2017-11-19	400	-20	380
2017-12-03	340	-40	300
2017-12-17	360	-40	320
2017-12-31	540	-40	500
			=XIRR(...)



date	income	expenses	
2017-01-01	40	-3000	-2960
2017-01-25	40	-50	-10
2017-02-12	80	-50	30
2017-02-14	100	-30	70
2017-03-04	100	-20	80
2017-04-23	160	-30	130
2017-05-07	140	-20	120
2017-05-21	140	-40	100
2017-06-04	80	-40	40
2017-06-19	180	-30	150
2017-07-16	360	-40	320
2017-08-27	160	-30	130
2017-09-24	240	-20	220
2017-10-21	420	-50	370
2017-11-19	400	-20	380
2017-12-03	340	-40	300
2017-12-17	360	-40	320
2017-12-31	540	-40	500
			=XIRR(...)



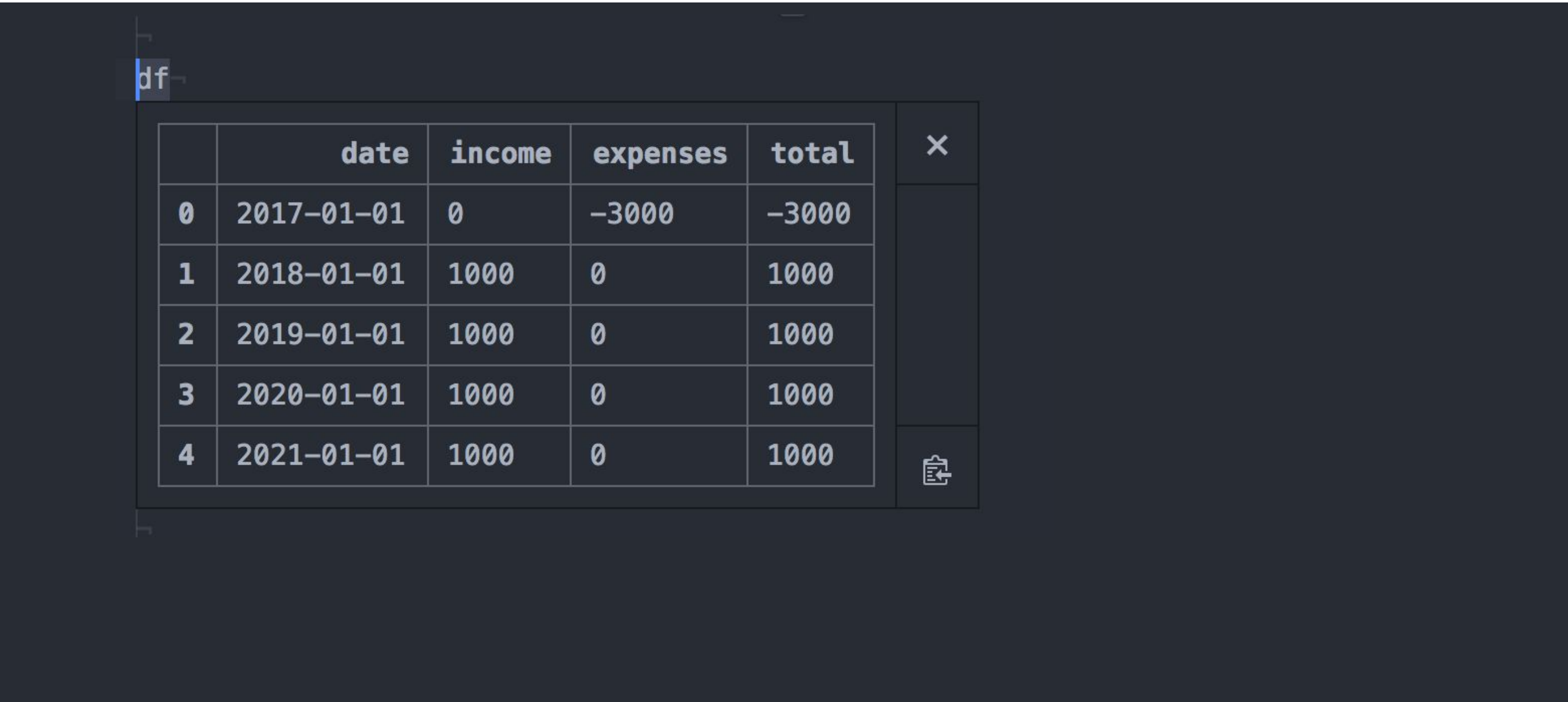
date	income	expenses	
2017-01-01	40	-3000	-2960
2017-01-25	40	-50	-10
2017-02-12	80	-50	30
2017-02-14	100	-30	70
2017-03-04	100	-20	80
2017-04-23	160	-30	130
2017-05-07	140	-20	120
2017-05-21	140	-40	100
2017-06-04	80	-40	40
2017-06-19	180	-30	150
2017-07-16	360	-40	320
2017-08-27	160	-30	130
2017-09-24	240	-20	220
2017-10-21	420	-50	370
2017-11-19	400	-20	380
2017-12-03	340	-40	300
2017-12-17	360	-40	320
2017-12-31	540	-40	500
			13.8%

why you shouldn’t use excel...

why you shouldn’t use excel...

```
import pandas as pd
from excel_functions import xirr, xnpv

df = pd.read_excel('data/irr.xlsx', sheet_name='regular')
```

```python
import pandas as pd
from excel_functions import xirr, xnpv

df = pd.read_excel('data/irr.xlsx', sheet_name='regular')
df['total'] = df.income + df.expenses
```



```python
df
```

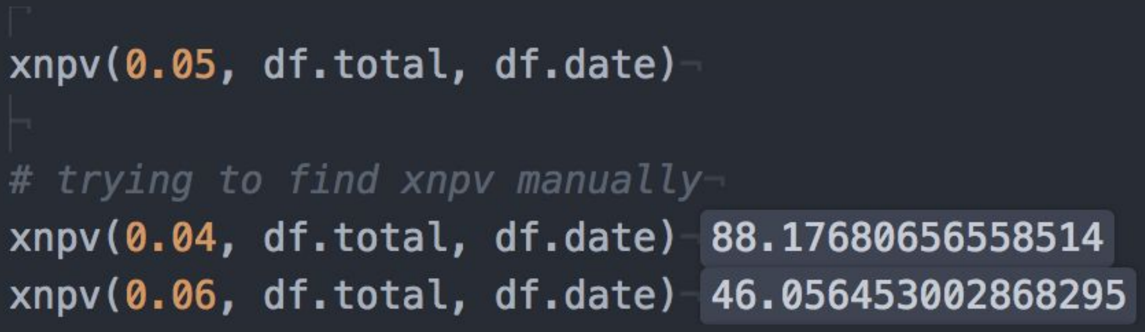
	date	income	expenses	total
0	2017-01-01	0	-3000	-3000
1	2018-01-01	1000	0	1000
2	2019-01-01	1000	0	1000
3	2020-01-01	1000	0	1000
4	2021-01-01	1000	0	1000

```python
def xnpv(rate, values, dates):
    '''Replicates the XNPV() function'''
    if rate <= -1.0:
        return float('inf')
    min_date = min(dates)
    return sum([
        value / (1 + rate)**((date - min_date).days / 365)
        for value, date
        in zip(values, dates)
    ])
```

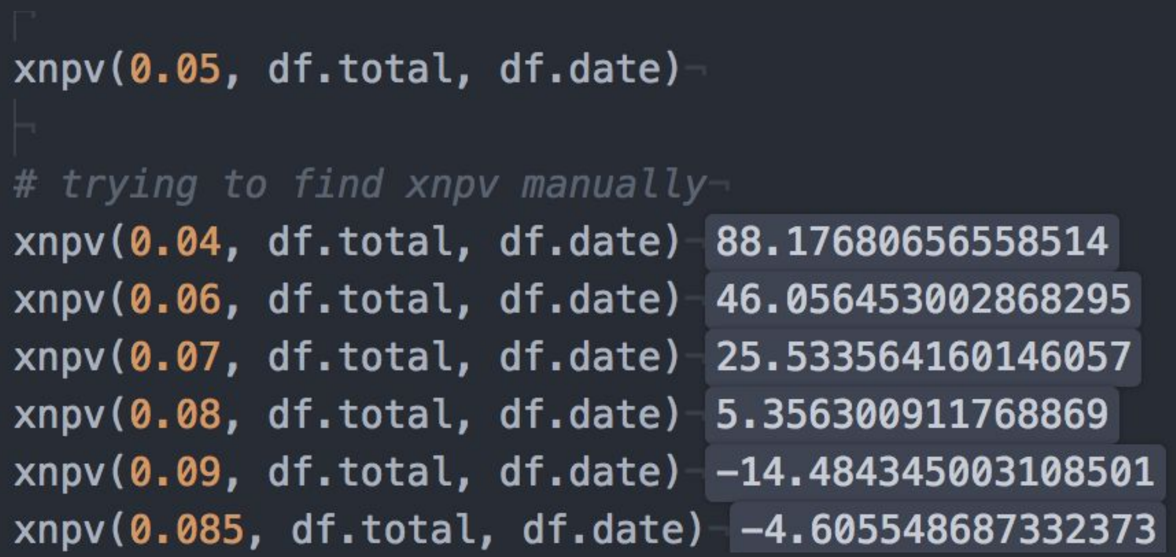
```python
def xnpv(rate, values, dates):
    '''Replicates the XNPV() function'''
    if rate <= -1.0:
        return float('inf')
    min_date = min(dates)
    return sum([
        value / (1 + rate)**((date - min_date).days / 365)
        for value, date
        in zip(values, dates)
    ])


xnpv(0.05, df.total, df.date)
```

```python
def xnpv(rate, values, dates):
    '''Replicates the XNPV() function'''
    if rate <= -1.0:
        return float('inf')
    min_date = min(dates)
    return sum([
        value / (1 + rate)**((date - min_date).days / 365)
        for value, date
        in zip(values, dates)
    ])


xnpv(0.05, df.total, df.date)
```

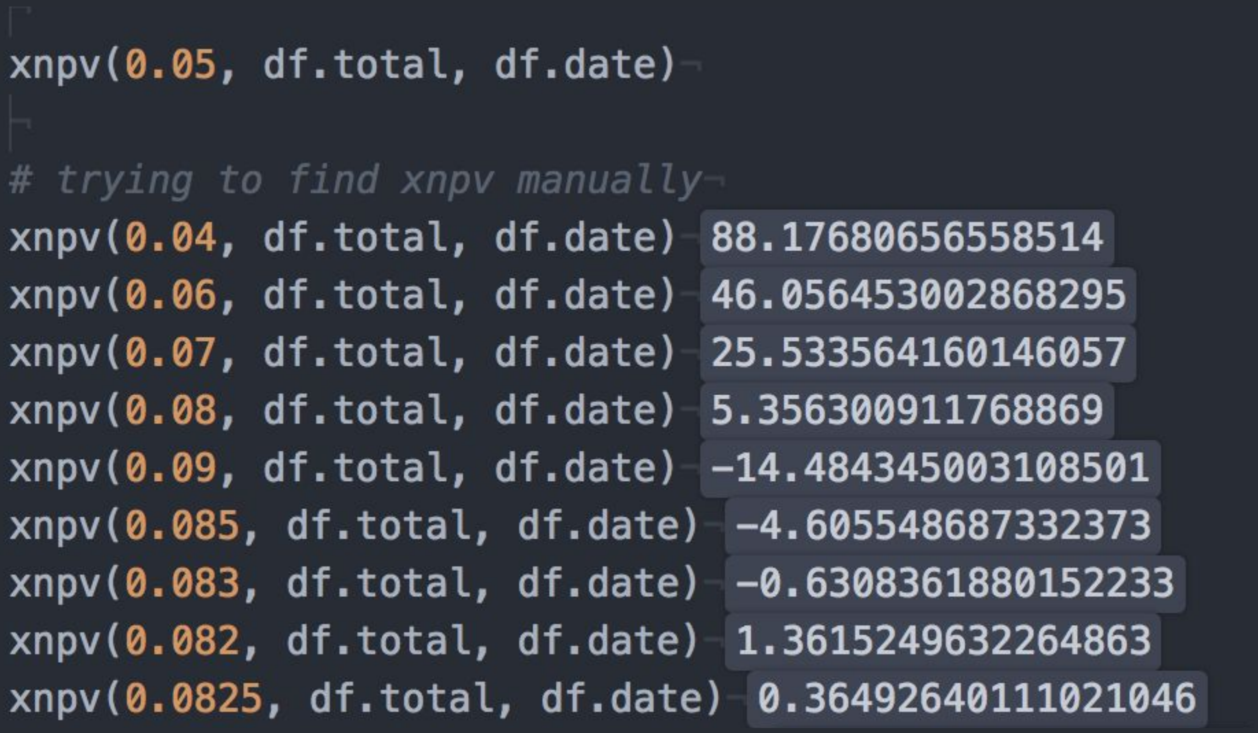
```
>>> 66.93430582852557
```



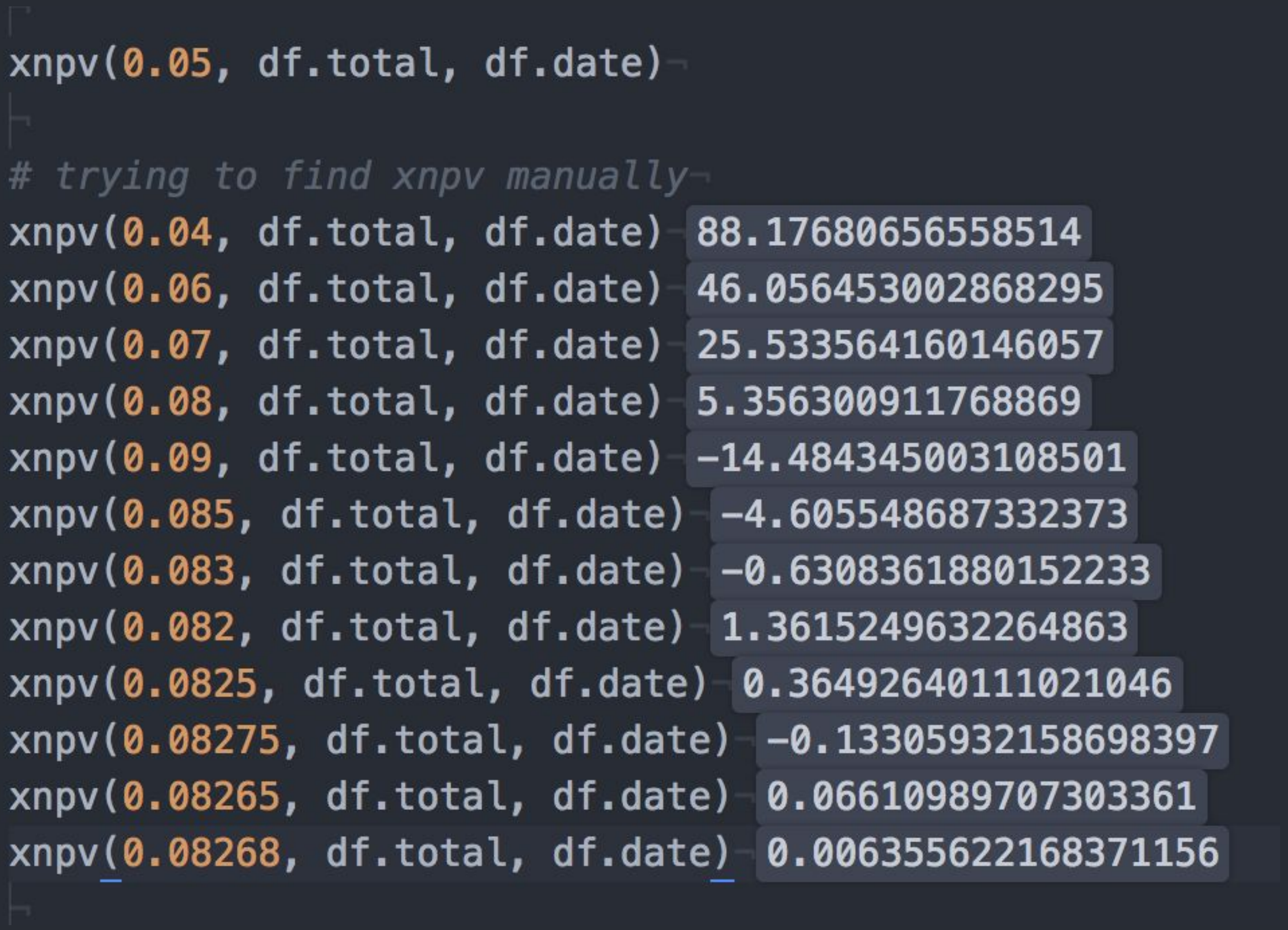
```
xnpv(0.05, df.total, df.date)

# trying to find xnpv manually
xnpv(0.04, df.total, df.date) 88.17680656558514
xnpv(0.06, df.total, df.date) 46.056453002868295
```



```
xnpv(0.05, df.total, df.date)

# trying to find xnpv manually
xnpv(0.04, df.total, df.date) 88.17680656558514
xnpv(0.06, df.total, df.date) 46.056453002868295
xnpv(0.07, df.total, df.date) 25.533564160146057
xnpv(0.08, df.total, df.date) 5.356300911768869
xnpv(0.09, df.total, df.date) -14.484345003108501
xnpv(0.085, df.total, df.date) -4.605548687332373
```



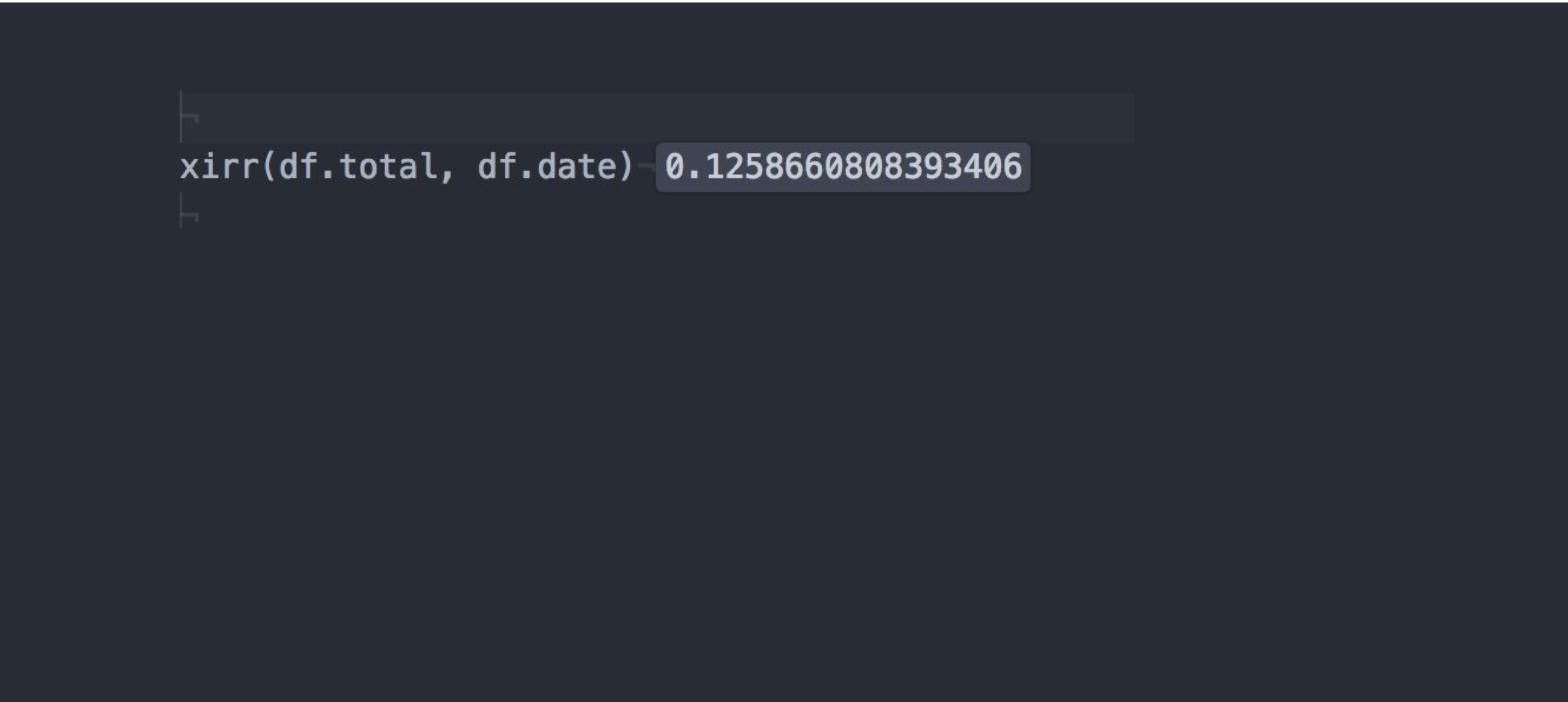
```
xnpv(0.05, df.total, df.date)

# trying to find xnpv manually
xnpv(0.04, df.total, df.date) 88.17680656558514
xnpv(0.06, df.total, df.date) 46.056453002868295
xnpv(0.07, df.total, df.date) 25.533564160146057
xnpv(0.08, df.total, df.date) 5.356300911768869
xnpv(0.09, df.total, df.date) -14.484345003108501
xnpv(0.085, df.total, df.date) -4.605548687332373
xnpv(0.083, df.total, df.date) -0.6308361880152233
xnpv(0.082, df.total, df.date) 1.3615249632264863
xnpv(0.0825, df.total, df.date) 0.3649264011021046
```



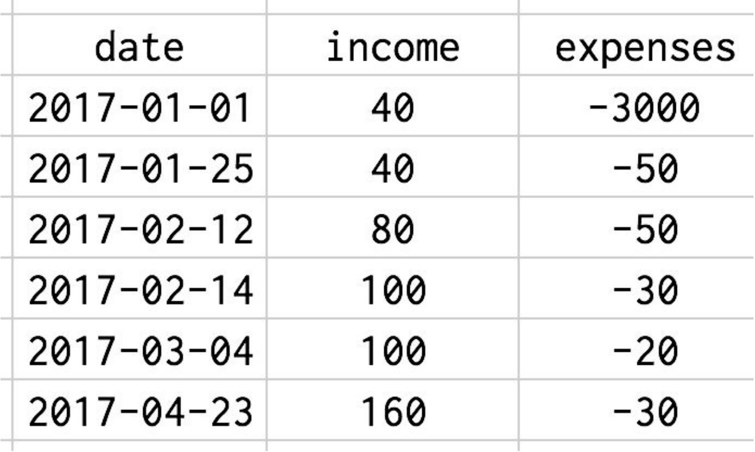
```
xnpv(0.05, df.total, df.date)

# trying to find xnpv manually
xnpv(0.04, df.total, df.date) 88.17680656558514
xnpv(0.06, df.total, df.date) 46.056453002868295
xnpv(0.07, df.total, df.date) 25.533564160146057
xnpv(0.08, df.total, df.date) 5.356300911768869
xnpv(0.09, df.total, df.date) -14.484345003108501
xnpv(0.085, df.total, df.date) -4.605548687332373
xnpv(0.083, df.total, df.date) -0.6308361880152233
xnpv(0.082, df.total, df.date) 1.3615249632264863
xnpv(0.0825, df.total, df.date) 0.36492640111021046
xnpv(0.08275, df.total, df.date) -0.13305932158698397
xnpv(0.08265, df.total, df.date) 0.06610989707303361
xnpv(0.08268, df.total, df.date) 0.006355622168371156
```

```python
from scipy.optimize import newton

def xirr(values, dates):
    '''Replicates the XIRR() function'''
    return newton(lambda r: xnpv(r, values, dates), 0)
```

```python
from scipy.optimize import newton

def xirr(values, dates):
    '''Replicates the XIRR() function'''
    return newton(lambda r: xnpv(r, values, dates), 0)
```

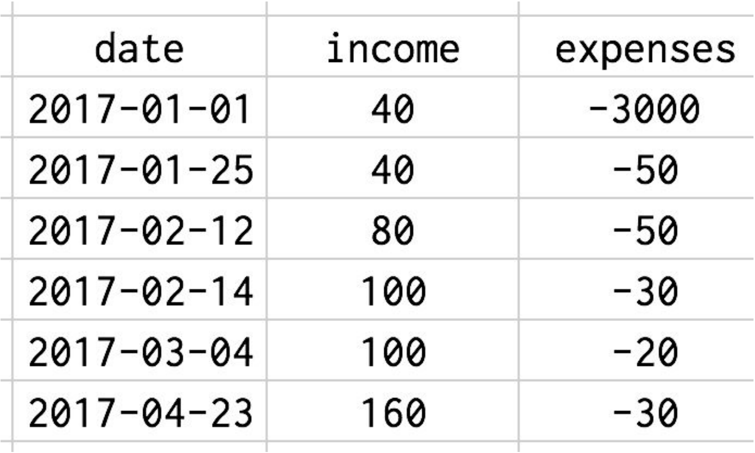


```
xirr(df.total, df.date) 0.1258660808393406
```



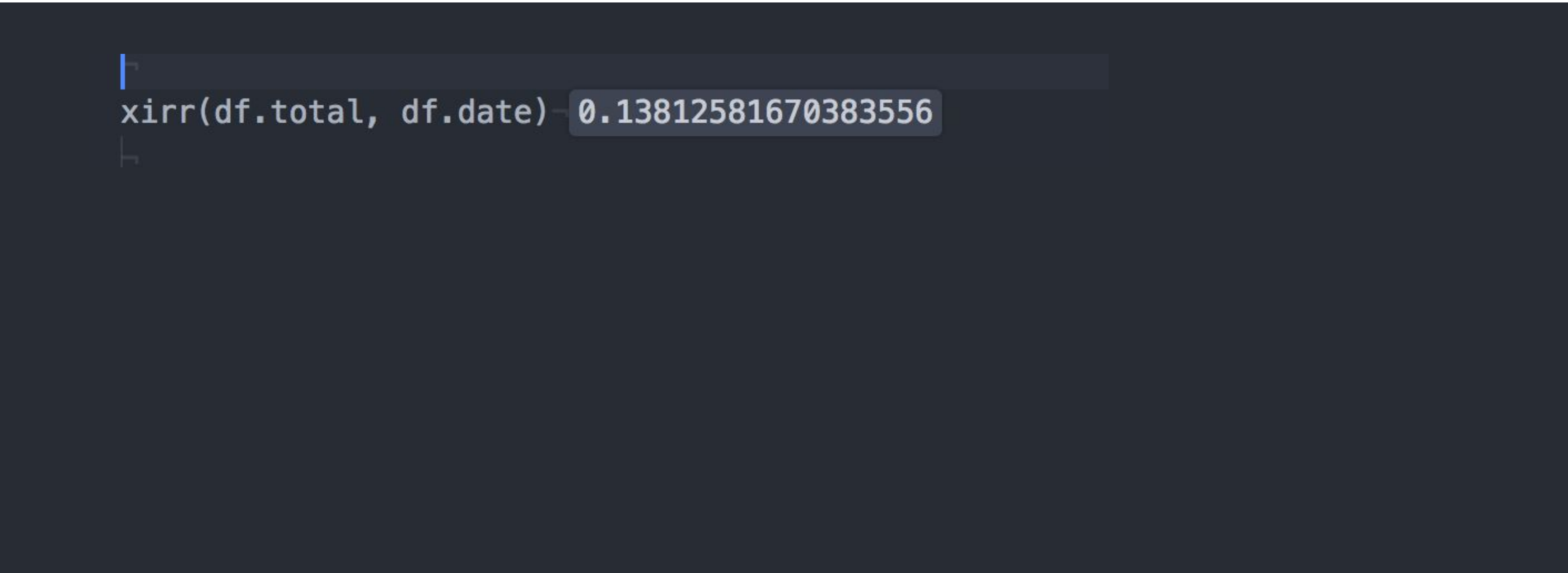
date	income	expenses
2017-01-01	40	-3000
2017-01-25	40	-50
2017-02-12	80	-50
2017-02-14	100	-30
2017-03-04	100	-20
2017-04-23	160	-30

```
df = pd.read_excel('data/irr.xlsx', sheet_name='irregular')
df['total'] = df.income + df.expenses
```

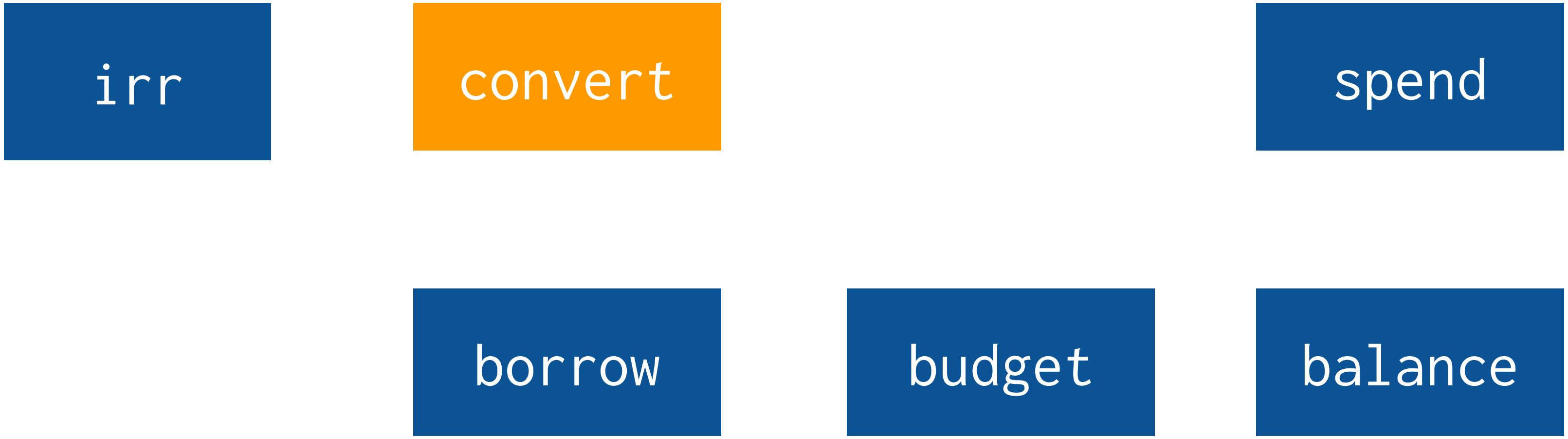


date	income	expenses
2017-01-01	40	-3000
2017-01-25	40	-50
2017-02-12	80	-50
2017-02-14	100	-30
2017-03-04	100	-20
2017-04-23	160	-30

```
df = pd.read_excel('data/irr.xlsx', sheet_name='irregular')
df['total'] = df.income + df.expenses
```



```
xirr(df.total, df.date) 0.1381258167038356
```



irr

convert

spend

borrow

budget

balance

open exchange rates

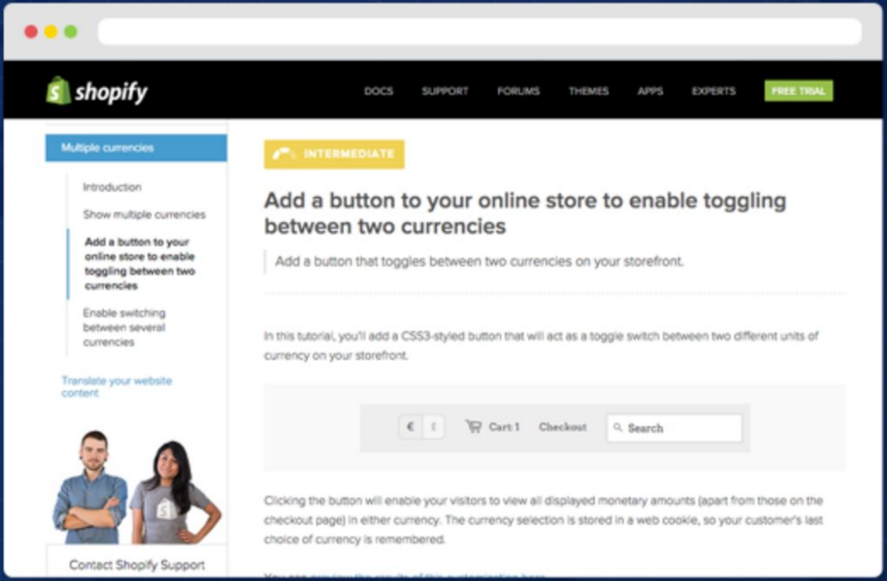
about us · documentation · help & support · api status · pricing | your dashboard

Our currency data API powers the Internet's most dynamic startups, brands and organisations.

Consistent, reliable exchange rate data and currency conversion for your business.

Flexible, fast, affordable – find out why more than 80,000 developers trust our API.

take a test drive or **Get Instant Access**



Add a button to your online store to enable toggling between two currencies

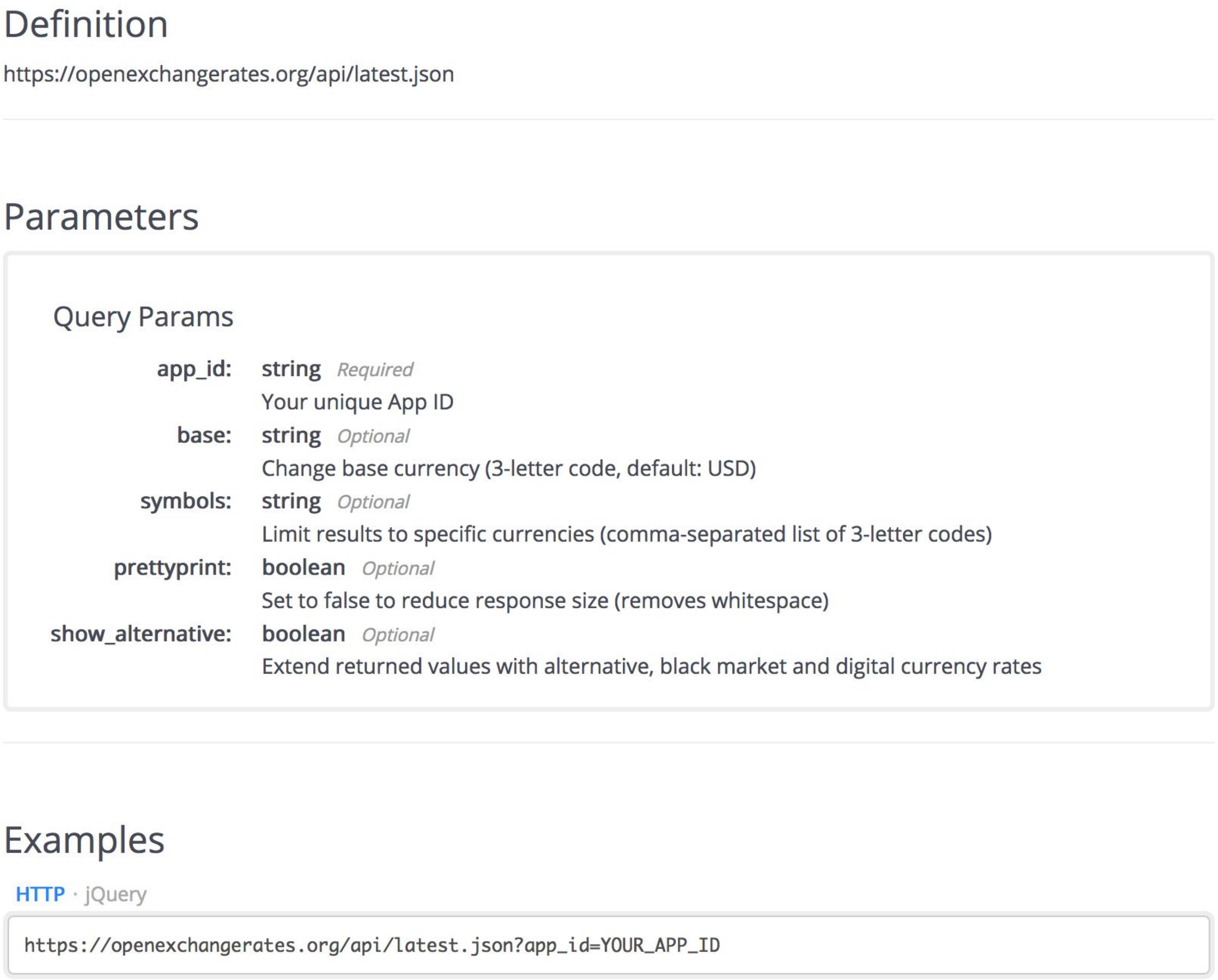
Add a button that toggles between two currencies on your storefront.

In this tutorial, you'll add a CSS3-styled button that will act as a toggle switch between two different units of currency on your storefront.

Clicking the button will enable your visitors to view all displayed monetary amounts (apart from those on the checkout page) in either currency. The currency selection is stored in a web cookie, so your customer's last choice of currency is remembered.

Powering seamless cross-currency payments at Shopify

```
import requests

requests.get('https://openexchangerates.org/api/latest.json')
```



Definition

https://openexchangerates.org/api/latest.json

Parameters

Query Params

app_id: **string** *Required*
Your unique App ID

base: **string** *Optional*
Change base currency (3-letter code, default: USD)

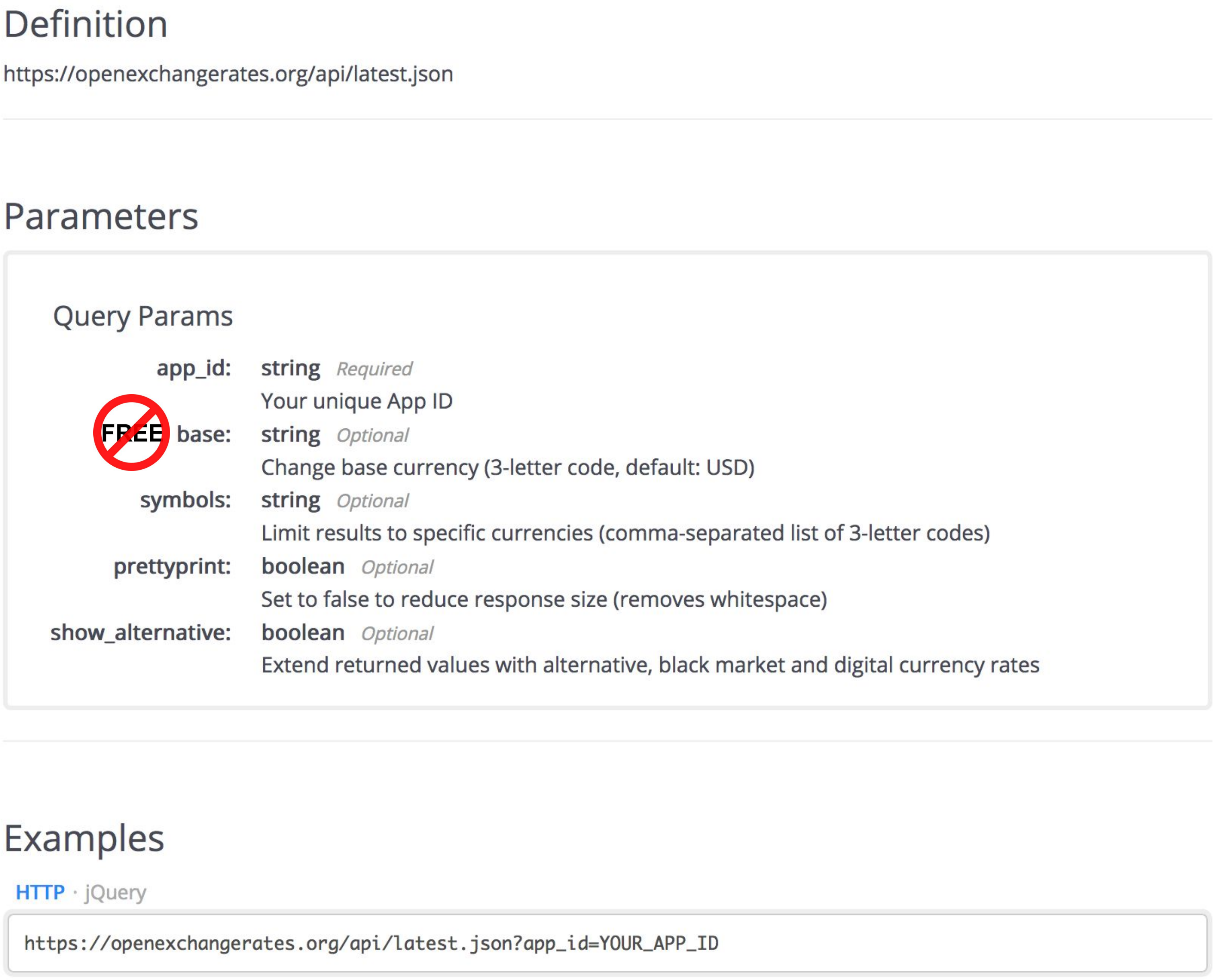
symbols: **string** *Optional*
Limit results to specific currencies (comma-separated list of 3-letter codes)

prettyprint: **boolean** *Optional*
Set to false to reduce response size (removes whitespace)

show_alternative: **boolean** *Optional*
Extend returned values with alternative, black market and digital currency rates

Examples

```
https://openexchangerates.org/api/latest.json?app_id=YOUR_APP_ID
```



Definition

https://openexchangerates.org/api/latest.json

Parameters

Query Params

app_id: **string** *Required*
Your unique App ID

🚫FREE **base:** **string** *Optional*
Change base currency (3-letter code, default: USD)

symbols: **string** *Optional*
Limit results to specific currencies (comma-separated list of 3-letter codes)

prettyprint: **boolean** *Optional*
Set to false to reduce response size (removes whitespace)

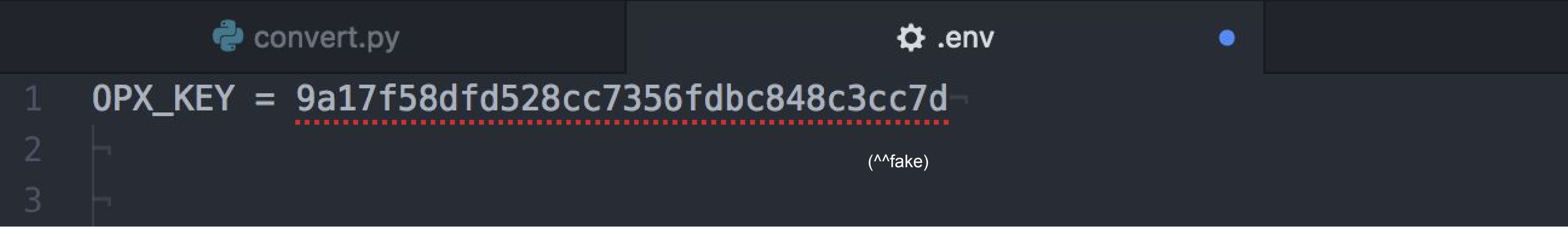
show_alternative: **boolean** *Optional*
Extend returned values with alternative, black market and digital currency rates

Examples

HTTP · jQuery

```
https://openexchangerates.org/api/latest.json?app_id=YOUR_APP_ID
```

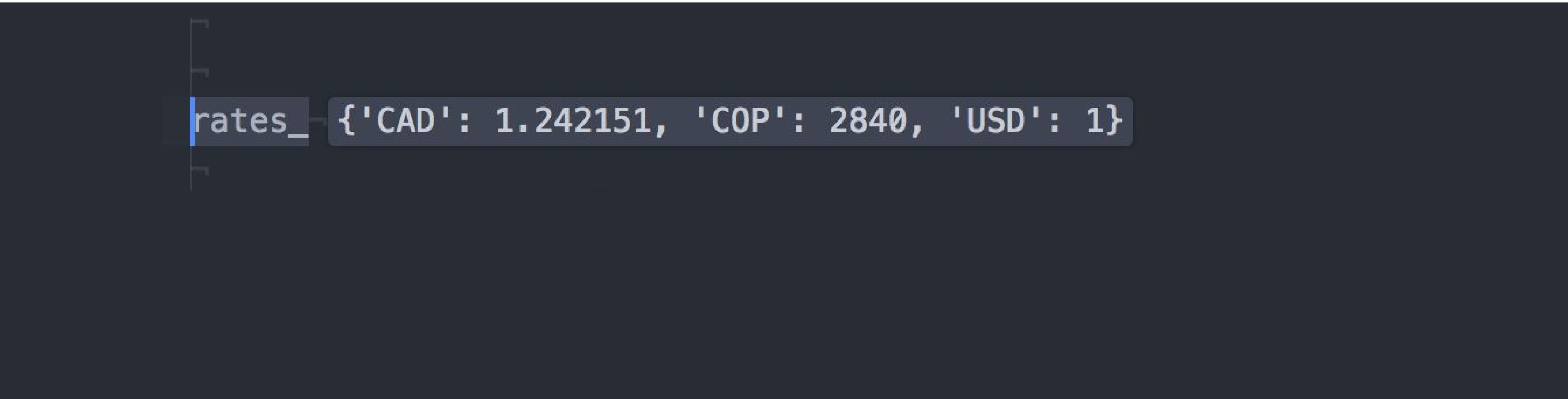
```
symbols = ['CAD', 'USD', 'COP']


r = requests.get(
    'https://openexchangerates.org/api/latest.json',
    params = {
        'app_id': API_KEY,
        'symbols': symbols,
        'show_alternative': 'true'
        }
    )
```



```
OPX_KEY = 9a17f58dfd528cc7356fdbc848c3cc7d
```

(^^fake)

```python
import os
import requests
from dotenv import load_dotenv, find_dotenv

load_dotenv(find_dotenv())

API_KEY = os.environ.get('OPX_KEY')
symbols = ['CAD', 'USD', 'COP']
symbols = ['CAD', 'USD', 'COP']


r = requests.get(
    'https://openexchangerates.org/api/latest.json',
    params = {
        'app_id': API_KEY,
        'symbols': symbols,
        'show_alternative': 'true'
        }
    )

rates_ = r.json()['rates']
```



```
rates_ {'CAD': 1.242151, 'COP': 2840, 'USD': 1}
```

```python
symbol_from = 'CAD'
symbol_to = 'COP'
value = 100

value * 1/rates_.get(symbol_from) * rates_.get(symbol_to)
```

```
symbol_from = 'CAD'
symbol_to = 'COP'
value = 100

value * 1/rates_.get(symbol_from) * rates_.get(symbol_to)
```

```
>>> 228635.65
```

```python
class CurrencyConverter:

    def __init__(self, symbols, API_KEY):

        self.API_KEY = API_KEY
        self.symbols = symbols
        self._symbols = ','.join([str(s) for s in symbols])

        r = requests.get(
            'https://openexchangerates.org/api/latest.json',
            params = {
                'app_id': self.API_KEY,
                'symbols': self._symbols,
                'show_alternative': 'true'
                }
            )

        self.rates_ = r.json()['rates']
        self.rates_['USD'] = 1
```

(CurrencyConverter continued…)

```
    def convert(self, value, symbol_from, symbol_to, round_output=True):

        try:
            x = value * 1/self.rates_.get(symbol_from) * self.rates_.get(symbol_to)
            if round_output:
                return round(x, 2)
            else:
                return x
        except TypeError:
            print('Unavailable or invalid symbol')
            return None
```

```
API_KEY = os.environ.get("OPX_KEY")
c = CurrencyConverter(['CAD', 'COP', 'BTC', 'ETH'], API_KEY)


c.convert(100, 'CAD', 'COP')
```

```python
API_KEY = os.environ.get("OPX_KEY")
c = CurrencyConverter(['CAD', 'COP', 'BTC', 'ETH'], API_KEY)


c.convert(100, 'CAD', 'COP')
```

```
>>> 228635.65
```

```
API_KEY = os.environ.get("OPX_KEY")
c = CurrencyConverter(['CAD', 'COP', 'DOGE'], API_KEY)


c.convert(100000, 'COP', 'DOGE')
```

```
API_KEY = os.environ.get("OPX_KEY")
c = CurrencyConverter(['CAD', 'COP', 'DOGE'], API_KEY)



c.convert(100000, 'COP', 'DOGE')
```

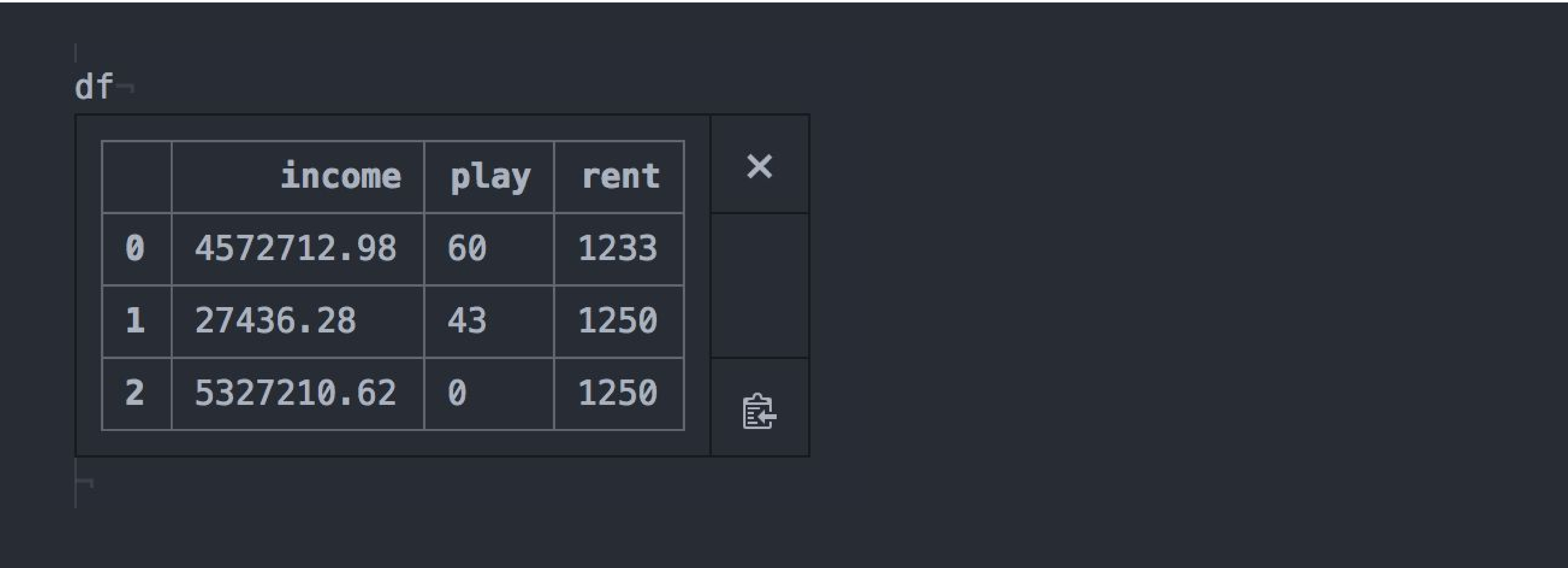
```
>>> 10599.63
```

```
API_KEY = os.environ.get("OPX_KEY")
c = CurrencyConverter(['CAD', 'COP', 'DOGE'], API_KEY)


c.convert(100000, 'COP', 'DOGE')
```

```
>>> 10599.63
```

```python
import pandas as pd

df = pd.DataFrame({
    'income': [2000, 12, 2330],
    'rent': [1233, 1250, 1250],
    'play': [60, 43, 0]
})

df['income'] = df['income'].apply(lambda x: c.convert(x, 'CAD', 'COP'))
```

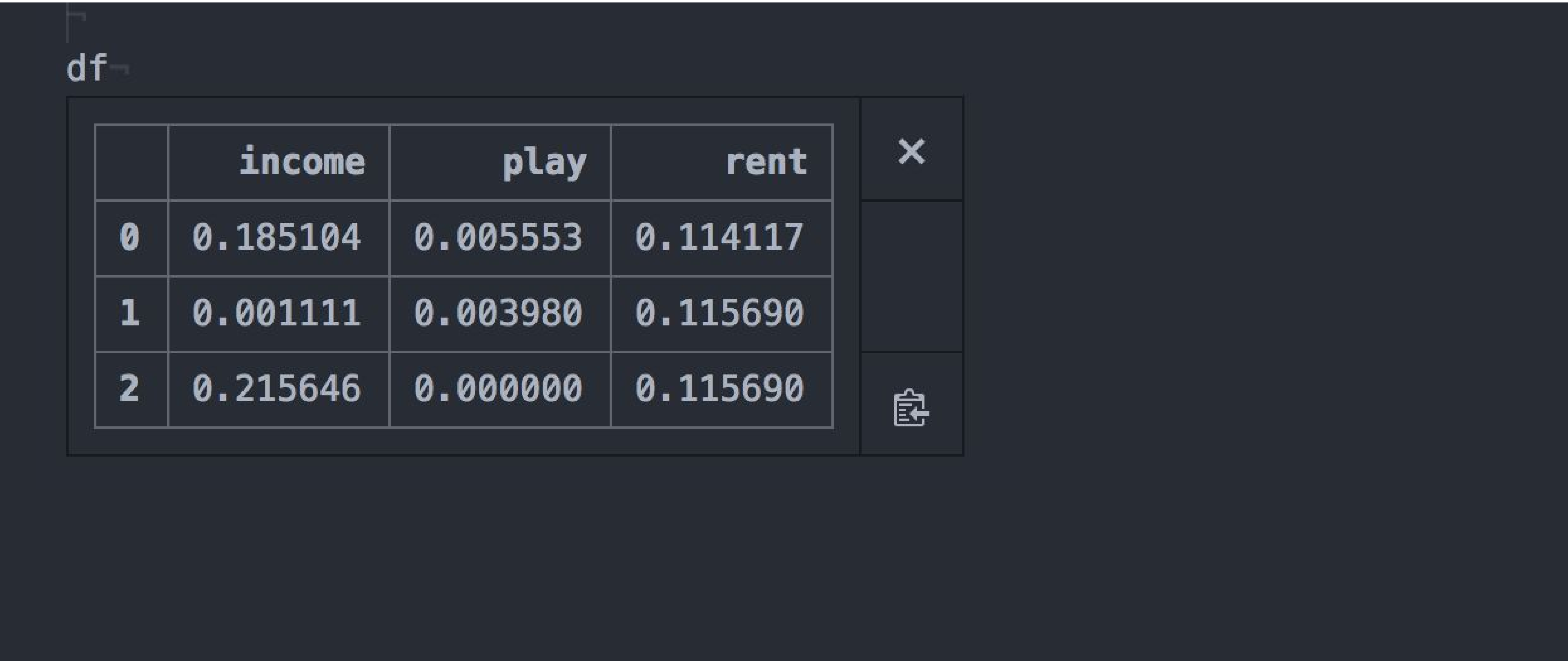
```python
import pandas as pd

df = pd.DataFrame({
    'income': [2000, 12, 2330],
    'rent': [1233, 1250, 1250],
    'play': [60, 43, 0]
})

df['income'] = df['income'].apply(lambda x: c.convert(x, 'CAD', 'COP'))
```



```
df
```

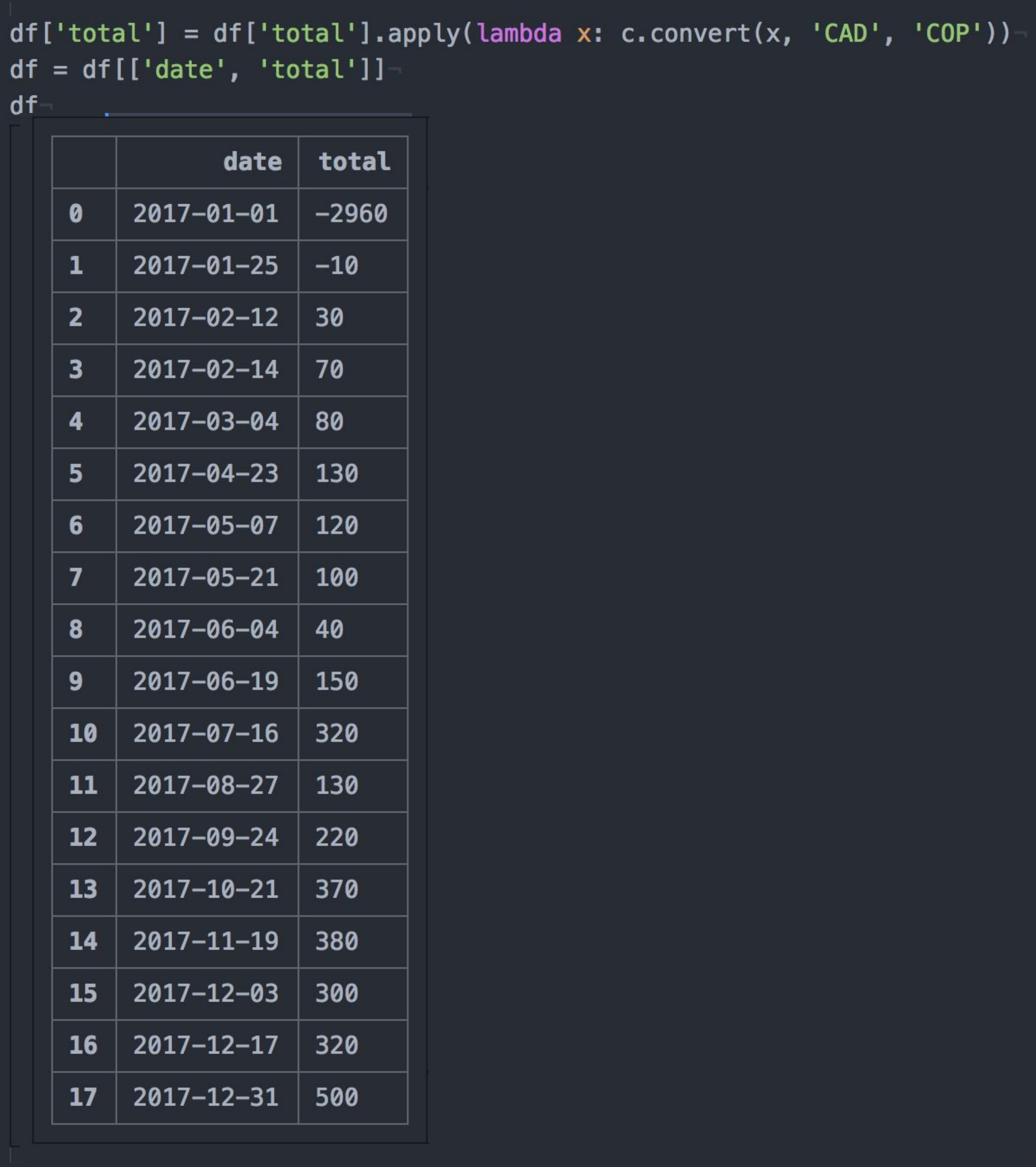
	income	play	rent
0	4572712.98	60	1233
1	27436.28	43	1250
2	5327210.62	0	1250

```
df.apply(lambda x: c.convert(x, 'CAD', 'BTC', round_output=False))
df.apply(lambda x: c.convert(x, 'CAD', 'BTC', round_output=False))
```



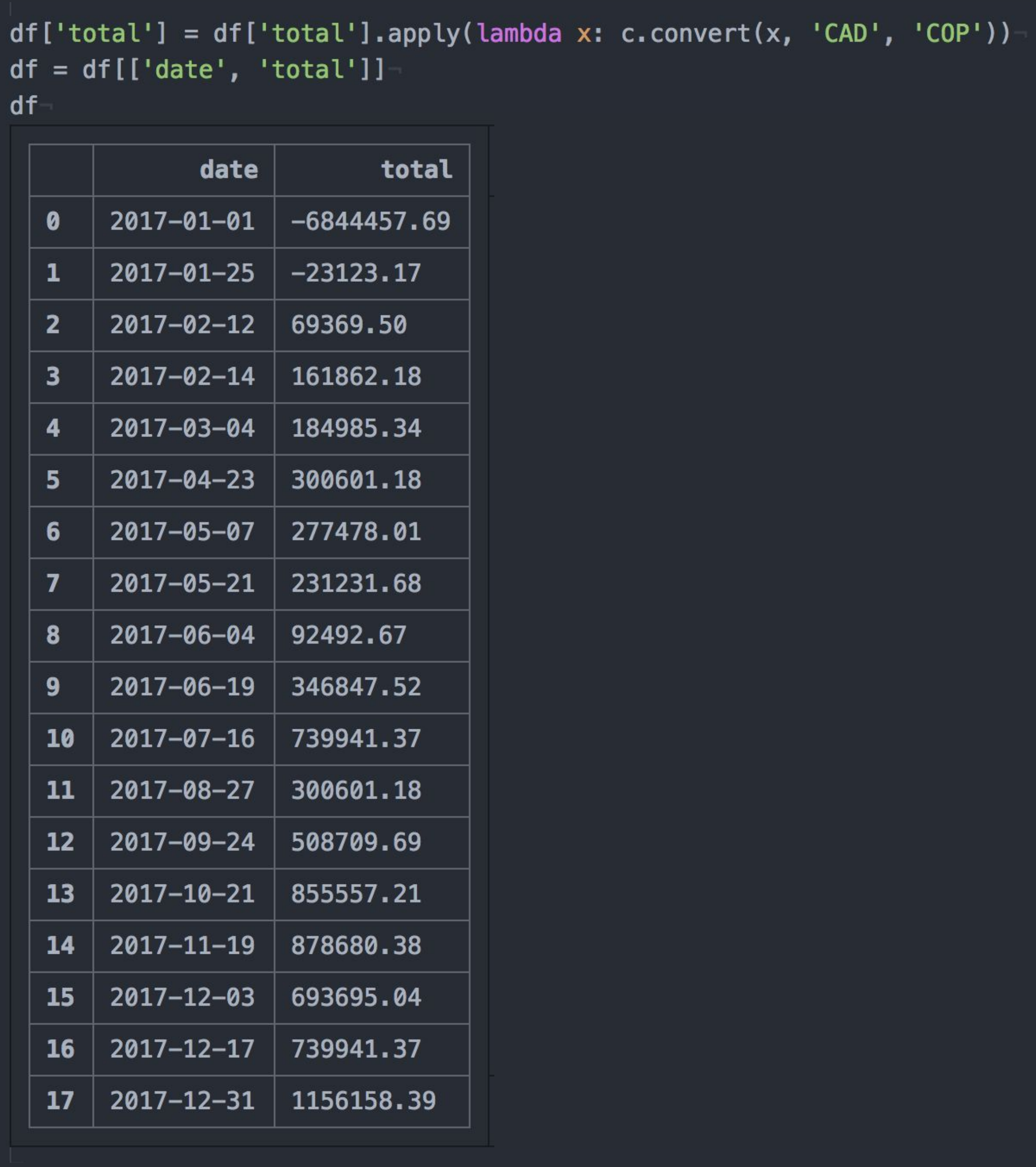
```
df
```

	income	play	rent
0	0.185104	0.005553	0.114117
1	0.001111	0.003980	0.115690
2	0.215646	0.000000	0.115690



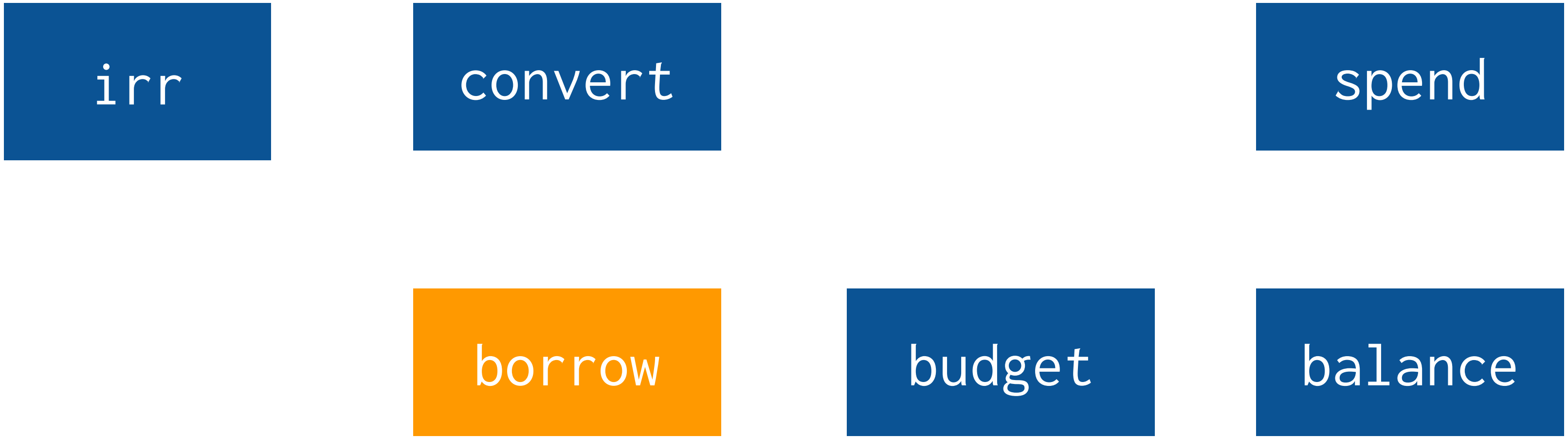
```
df['total'] = df['total'].apply(lambda x: c.convert(x, 'CAD', 'COP'))
df = df[['date', 'total']]
df
```

	date	total
0	2017-01-01	-2960
1	2017-01-25	-10
2	2017-02-12	30
3	2017-02-14	70
4	2017-03-04	80
5	2017-04-23	130
6	2017-05-07	120
7	2017-05-21	100
8	2017-06-04	40
9	2017-06-19	150
10	2017-07-16	320
11	2017-08-27	130
12	2017-09-24	220
13	2017-10-21	370
14	2017-11-19	380
15	2017-12-03	300
16	2017-12-17	320
17	2017-12-31	500



```
df['total'] = df['total'].apply(lambda x: c.convert(x, 'CAD', 'COP'))
df = df[['date', 'total']]
df
```

	date	total
0	2017-01-01	-6844457.69
1	2017-01-25	-23123.17
2	2017-02-12	69369.50
3	2017-02-14	161862.18
4	2017-03-04	184985.34
5	2017-04-23	300601.18
6	2017-05-07	277478.01
7	2017-05-21	231231.68
8	2017-06-04	92492.67
9	2017-06-19	346847.52
10	2017-07-16	739941.37
11	2017-08-27	300601.18
12	2017-09-24	508709.69
13	2017-10-21	855557.21
14	2017-11-19	878680.38
15	2017-12-03	693695.04
16	2017-12-17	739941.37
17	2017-12-31	1156158.39



irr

convert

spend

borrow

budget

balance

3000.00 USD

3000.00 USD

↓

8,520,000.00 COP

3000.00 USD

↓

8,520,000.00 COP

Banco de Bogotá

Banco AV Villas

Bancolombia

8,520,000.00 COP

Banco de Bogotá	Banco AV Villas	Bancolombia
14 months	20 months	8 months
5.75%	3.99%	8.99%

Banco de Bogotá	Banco AV Villas	Bancolombia
14 months	20 months	8 months
5.75%	3.99%	8.99%

Time is money, money is power,
power is pizza and pizza is knowledge

```python
import pandas as pd
import numpy as np
import datetime

loan = 8520000.00
rate = 0.05
term = 120
```

```python
import pandas as pd
import numpy as np
import datetime

loan = 8520000.00
rate = 0.05
term = 120
```

$$P = \frac{r(PV)}{1-(1+r)^{-n}}$$

$P = Payment$

$PV = Present\ Value$

$r = rate\ per\ period$

$n = number\ of\ periods$

```python
import pandas as pd
import numpy as np
import datetime

loan = 8520000.00
rate = 0.05
term = 120
```

$$P = \frac{r(PV)}{1-(1+r)^{-n}}$$

$$
\begin{aligned}
&P = Payment\\
&PV = Present\ Value\\
&r = rate\ per\ period\\
&n = number\ of\ periods
\end{aligned}
$$

```python
payment = loan * (rate / 12) / (1 - (1 + (rate / 12))**(-term))
```

```
import pandas as pd
import numpy as np
import datetime

loan = 8520000.00
rate = 0.05
term = 120
```

$$P = \frac{r(PV)}{1 - (1 + r)^{-n}}$$

$$P = Payment$$
$$PV = Present\ Value$$
$$r = rate\ per\ period$$
$$n = number\ of\ periods$$

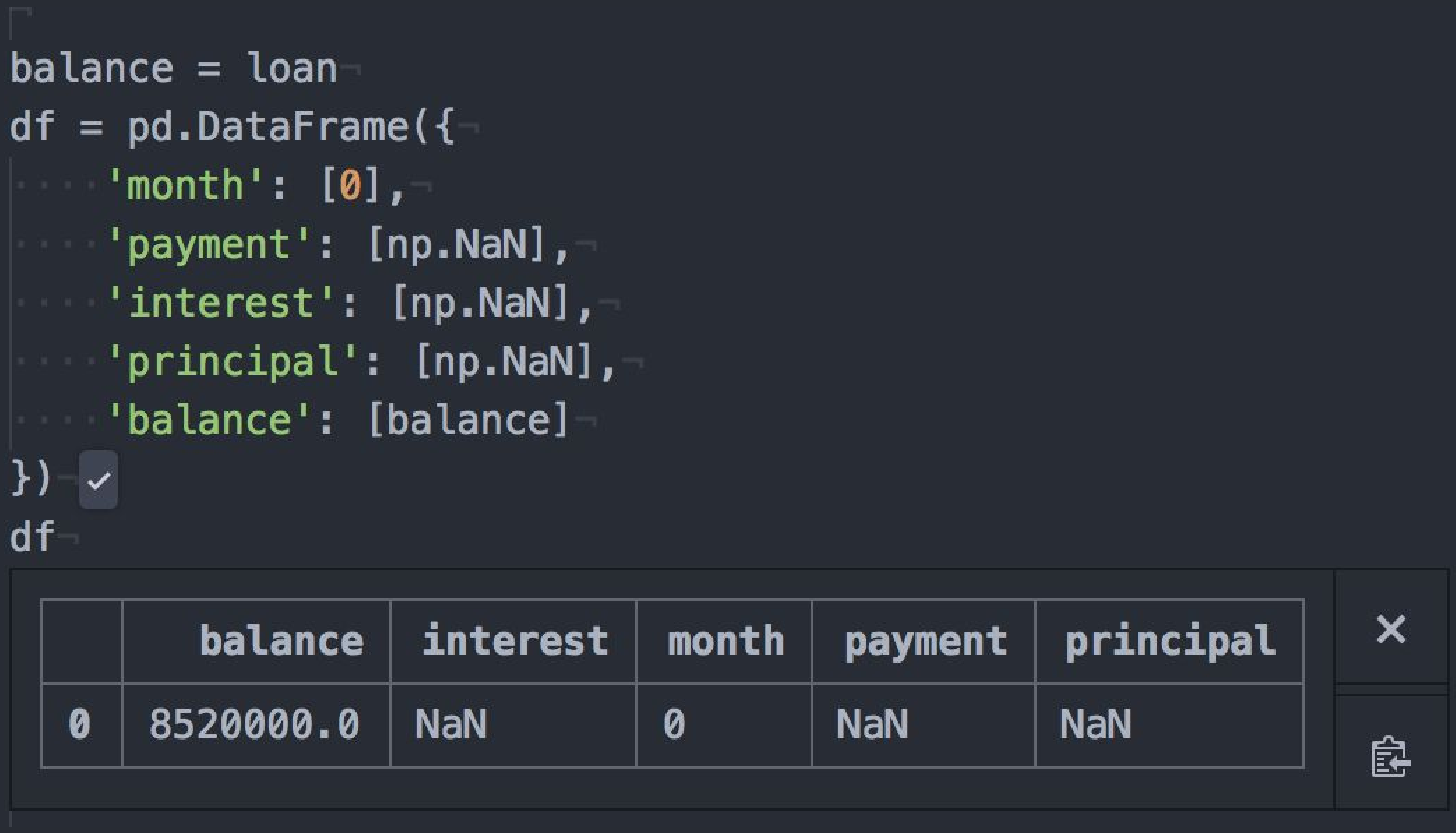
```
payment = loan * (rate / 12) / (1 - (1 + (rate / 12))**(-term))
```

```
>>> 80317.9562517743
```

```
import pandas as pd
import numpy as np
import datetime

loan = 8520000.00
rate = 0.05
term = 120
```

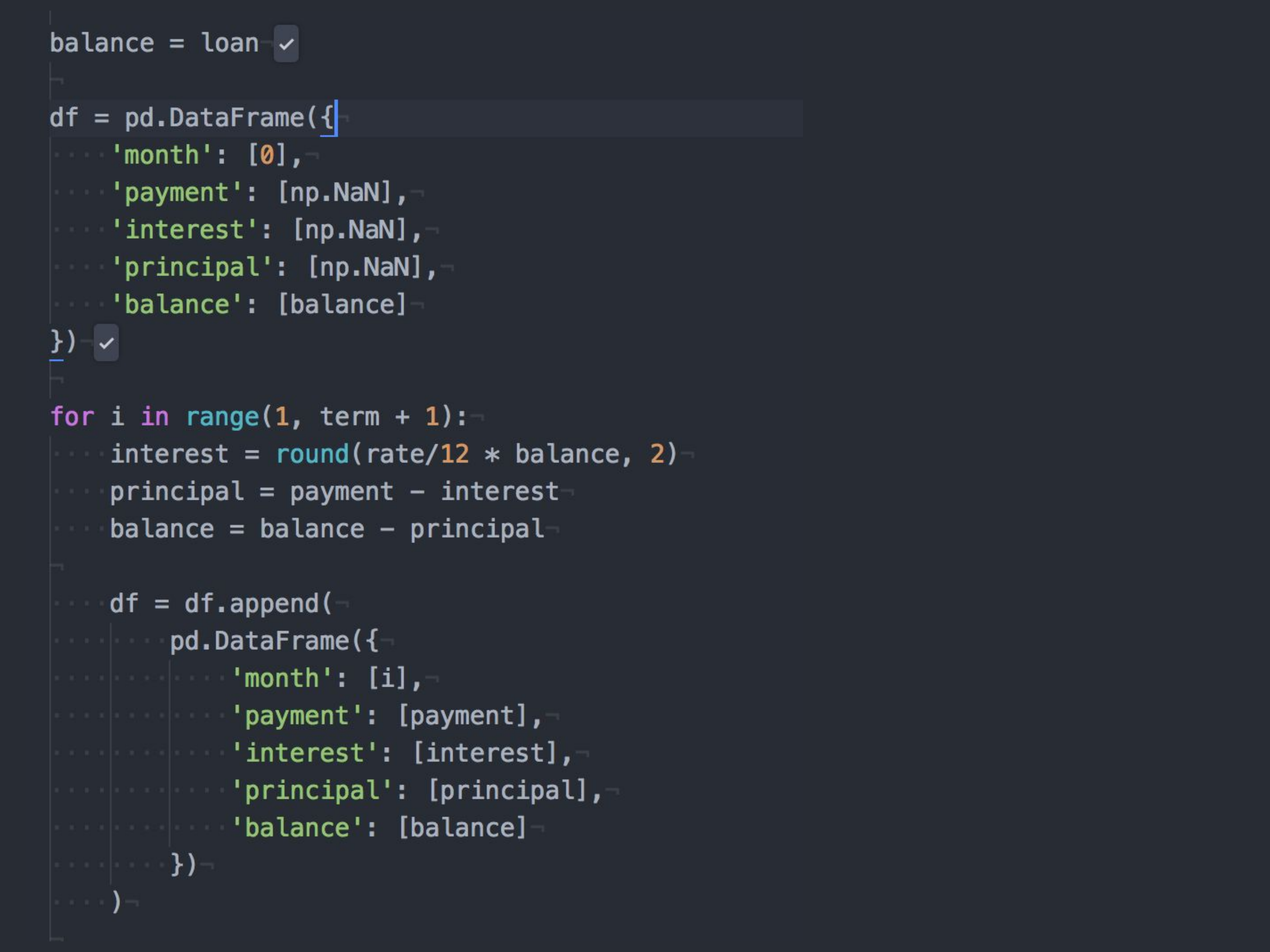
```
payment = round(-np.pmt(rate/12, term, loan), 2)
```

```
>>> 80317.96
```



```python
balance = loan
df = pd.DataFrame({
    'month': [0],
    'payment': [np.NaN],
    'interest': [np.NaN],
    'principal': [np.NaN],
    'balance': [balance]
})
df
```

	balance	interest	month	payment	principal
0	8520000.0	NaN	0	NaN	NaN

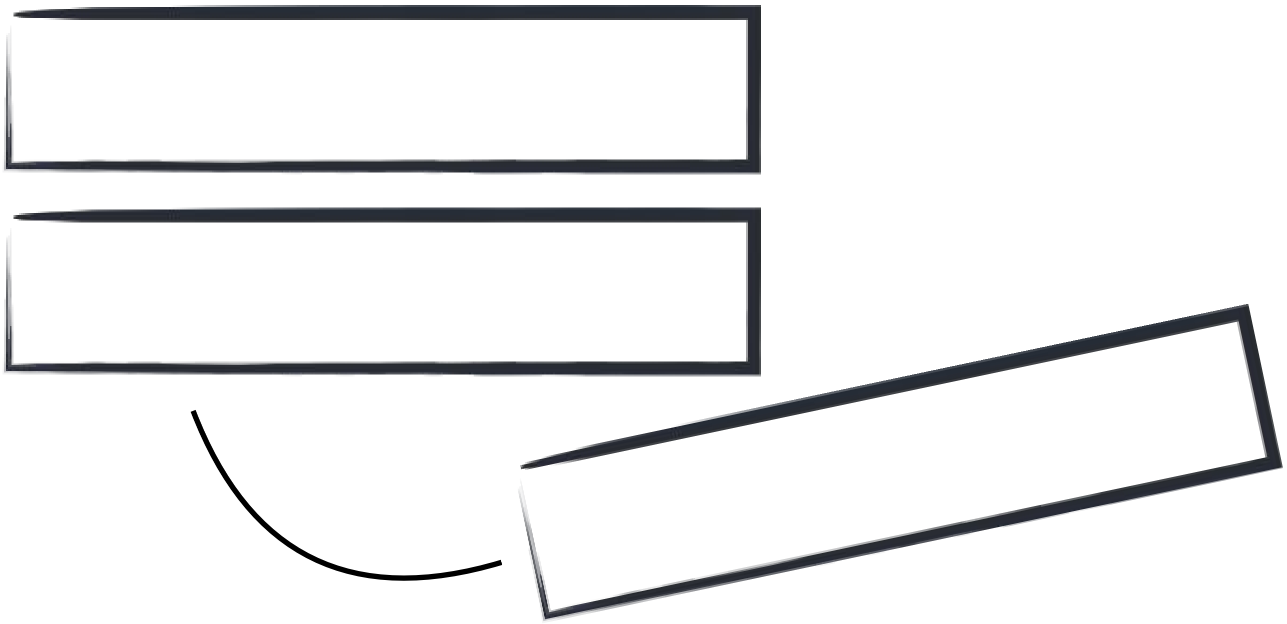


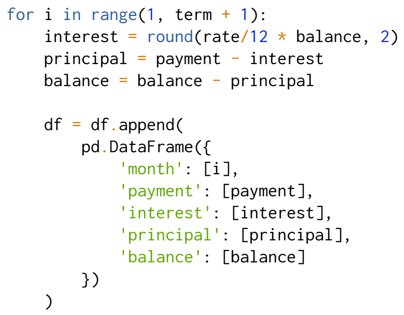
```python
balance = loan

df = pd.DataFrame({
    'month': [0],
    'payment': [np.NaN],
    'interest': [np.NaN],
    'principal': [np.NaN],
    'balance': [balance]
})

for i in range(1, term + 1):
    interest = round(rate/12 * balance, 2)
    principal = payment - interest
    balance = balance - principal

    df = df.append(
        pd.DataFrame({
            'month': [i],
            'payment': [payment],
            'interest': [interest],
            'principal': [principal],
            'balance': [balance]
        })
    )
```

```
for i in range(1, term + 1):
    interest = round(rate/12 * balance, 2)
    principal = payment - interest
    balance = balance - principal

    df = df.append(
        pd.DataFrame({
            'month': [i],
            'payment': [payment],
            'interest': [interest],
            'principal': [principal],
            'balance': [balance]
        })
    )
```

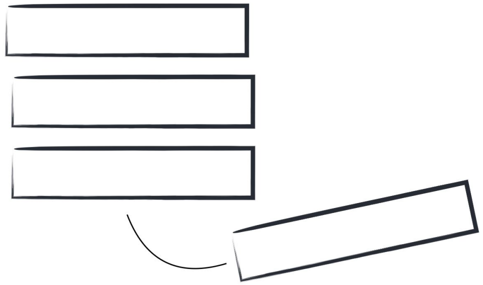
```
for i in range(1, term + 1):
    interest = round(rate/12 * balance, 2)
    principal = payment - interest
    balance = balance - principal

    df = df.append(
        pd.DataFrame({
            'month': [i],
            'payment': [payment],
            'interest': [interest],
            'principal': [principal],
            'balance': [balance]
        })
    )
```

```python
for i in range(1, term + 1):
    interest = round(rate/12 * balance, 2)
    principal = payment - interest
    balance = balance - principal

    df = df.append(
        pd.DataFrame({
            'month': [i],
            'payment': [payment],
            'interest': [interest],
            'principal': [principal],
            'balance': [balance]
        })
    )
```





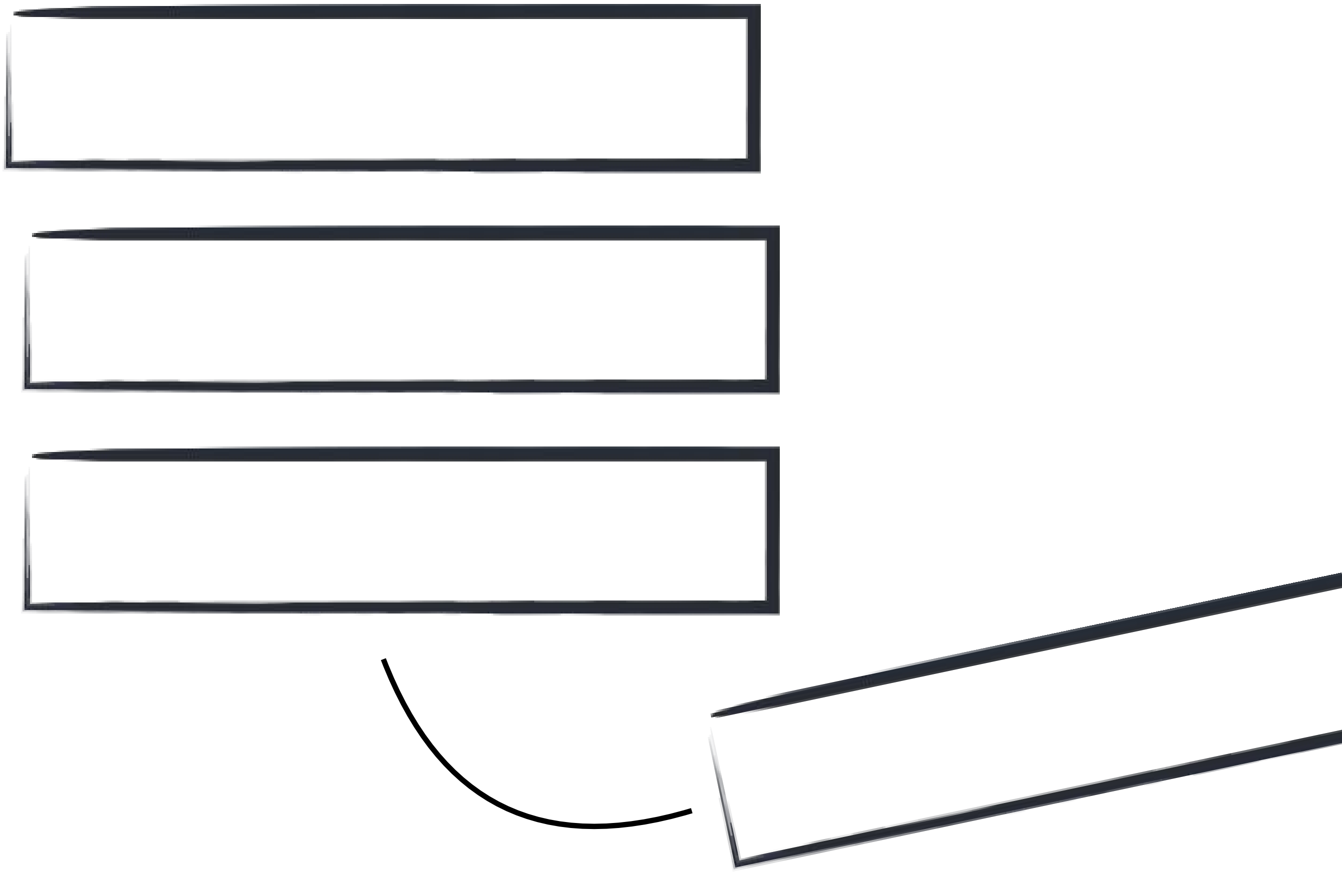
```
for i in range(1, term + 1):
    interest = round(rate/12 * balance, 2)
    principal = payment - interest
    balance = balance - principal

    df = df.append(
        pd.DataFrame({
            'month': [i],
            'payment': [payment],
            'interest': [interest],
            'principal': [principal],
            'balance': [balance]
        })
    )
```

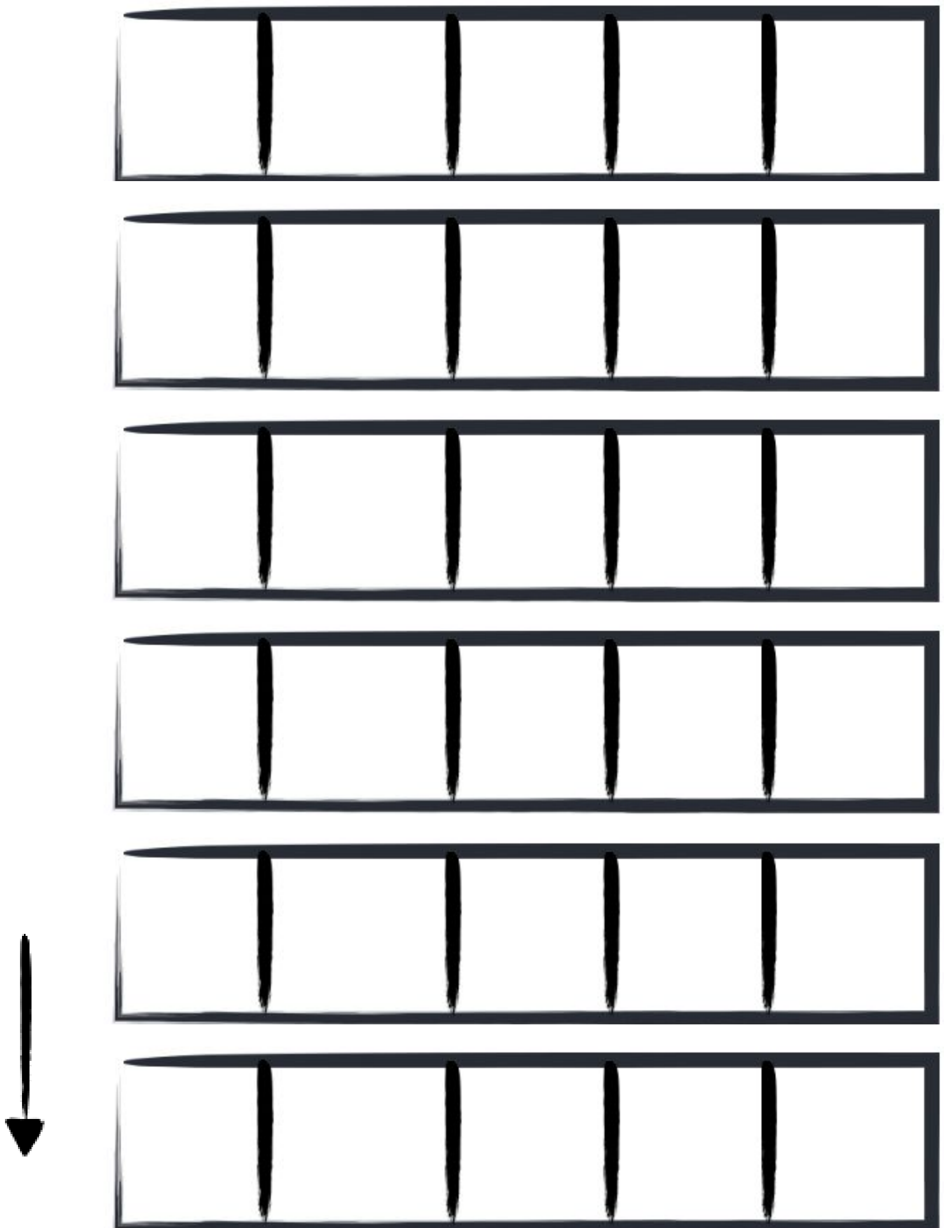


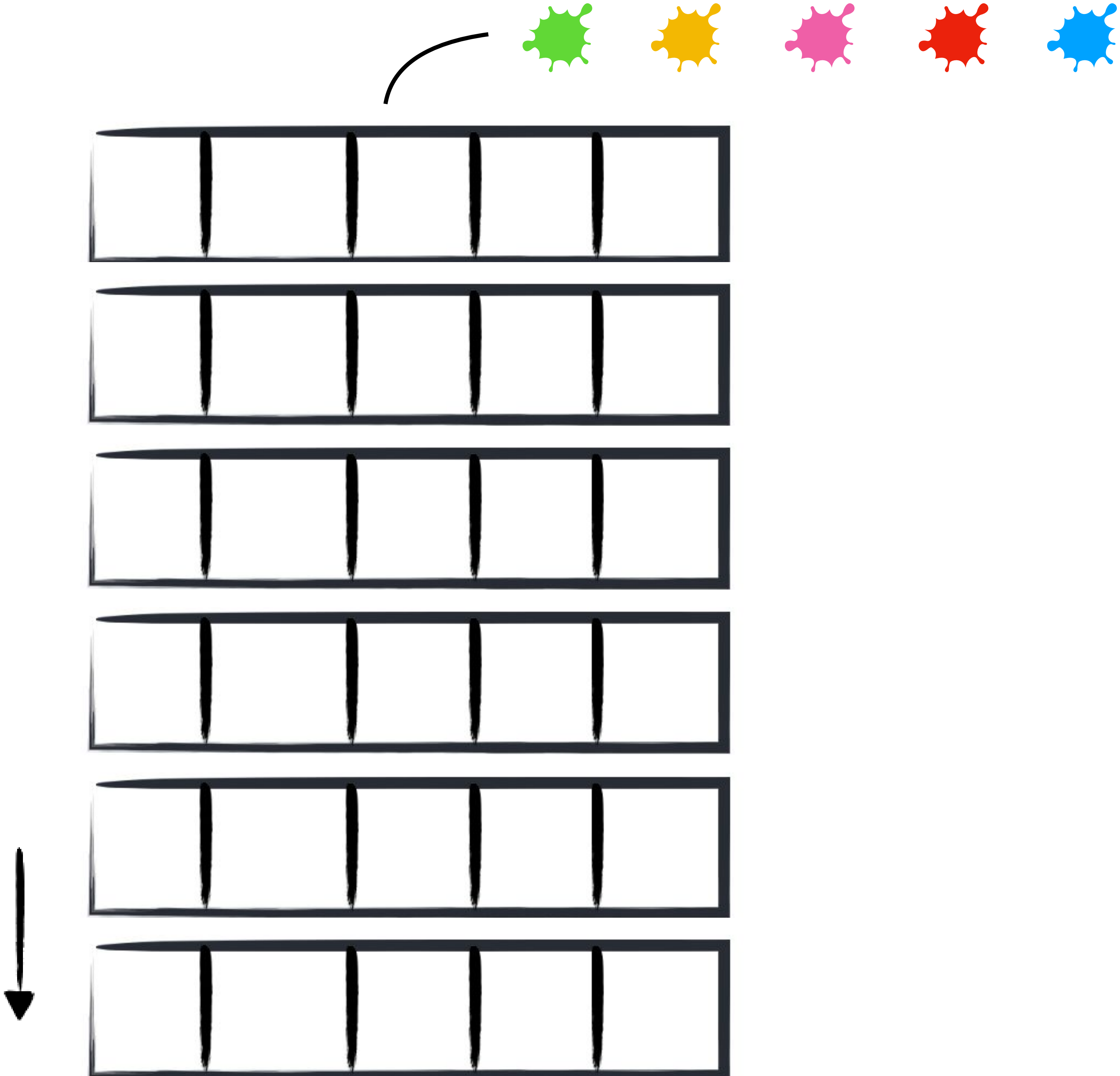
```
%%timeit
bad_way()
```

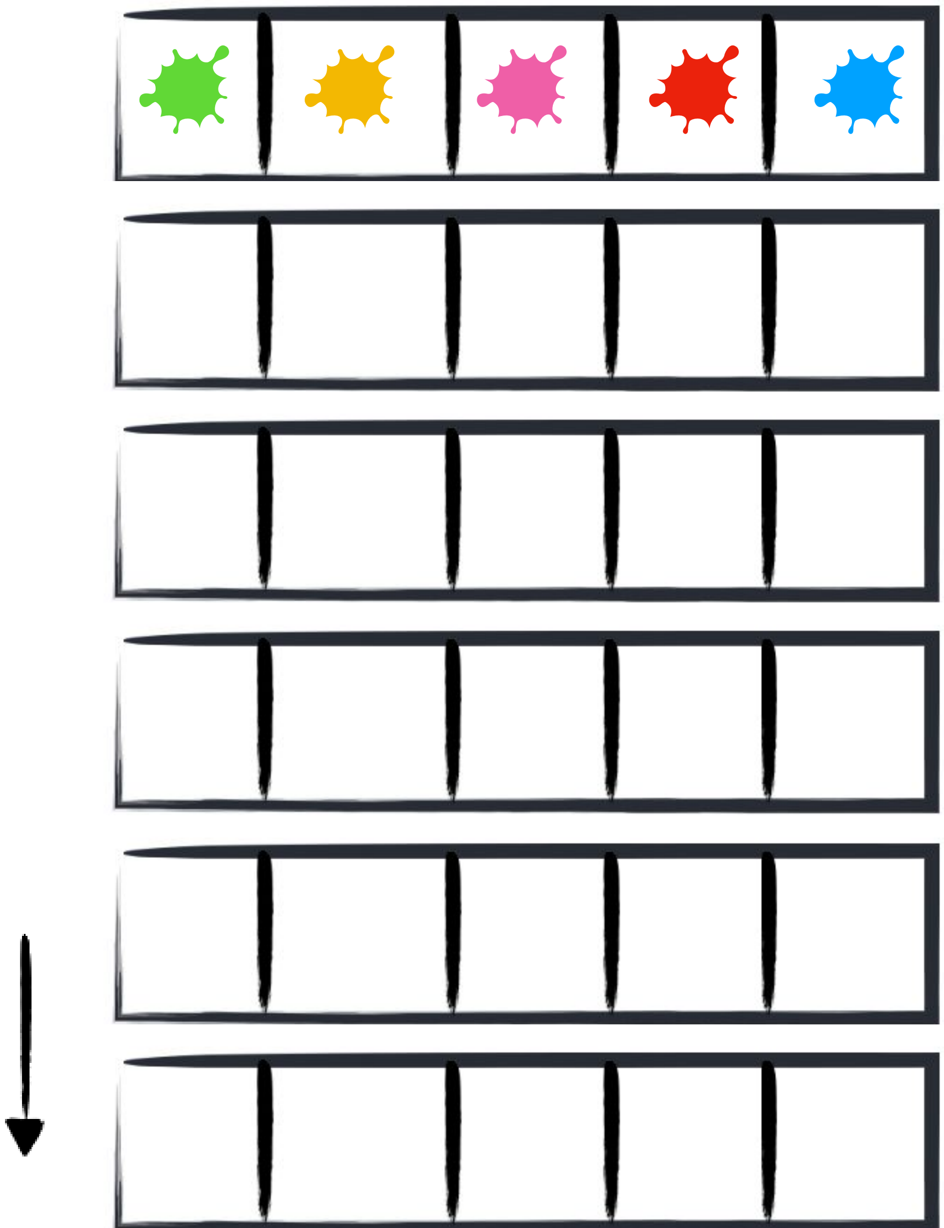
```
>>> 169 ms ± 7.48 ms per loop
>>> (mean ± std. dev. of 7 runs, 10 loops each)
```

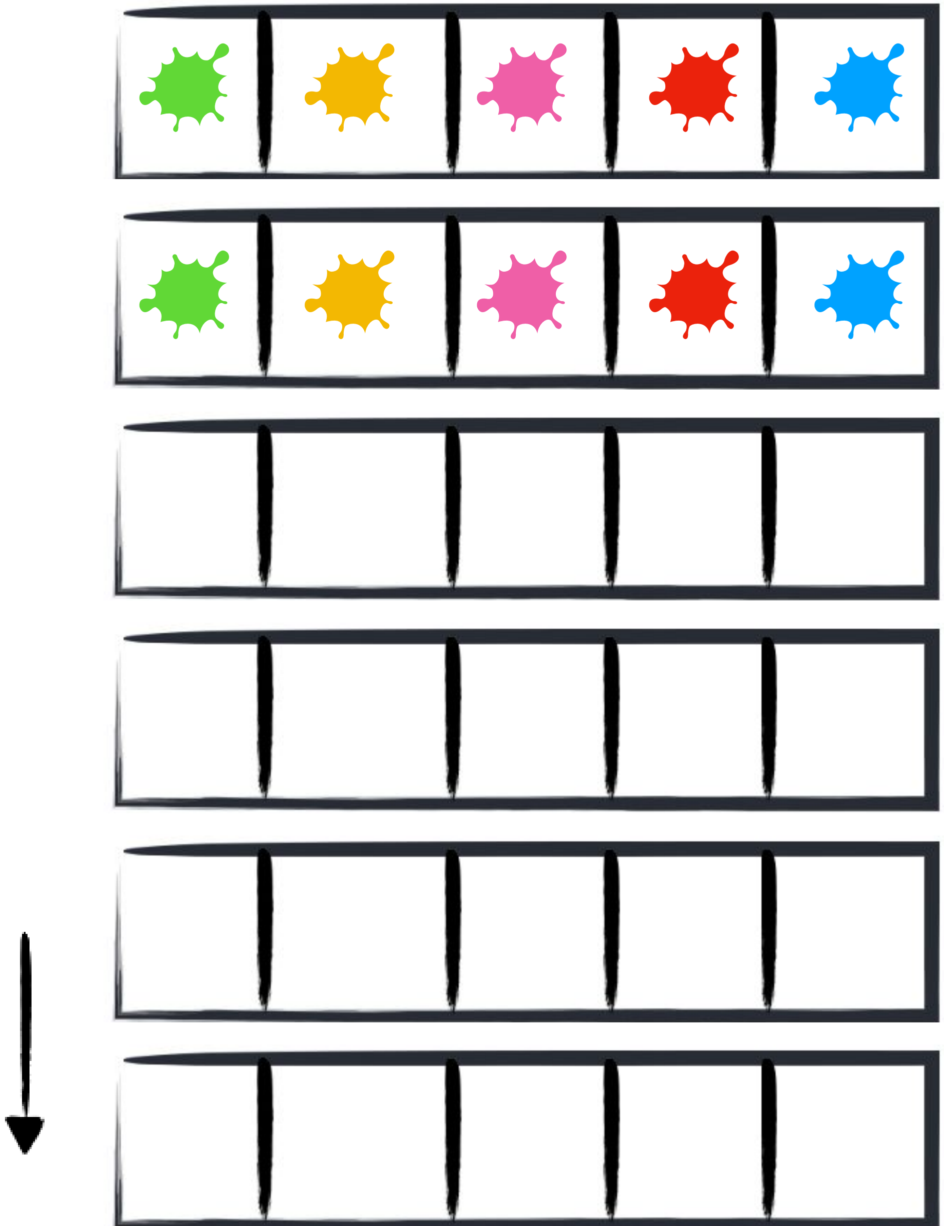
```
>>> 169 ms ± 7.48 ms per loop
>>> (mean ± std. dev. of 7 runs, 10 loops each)
```

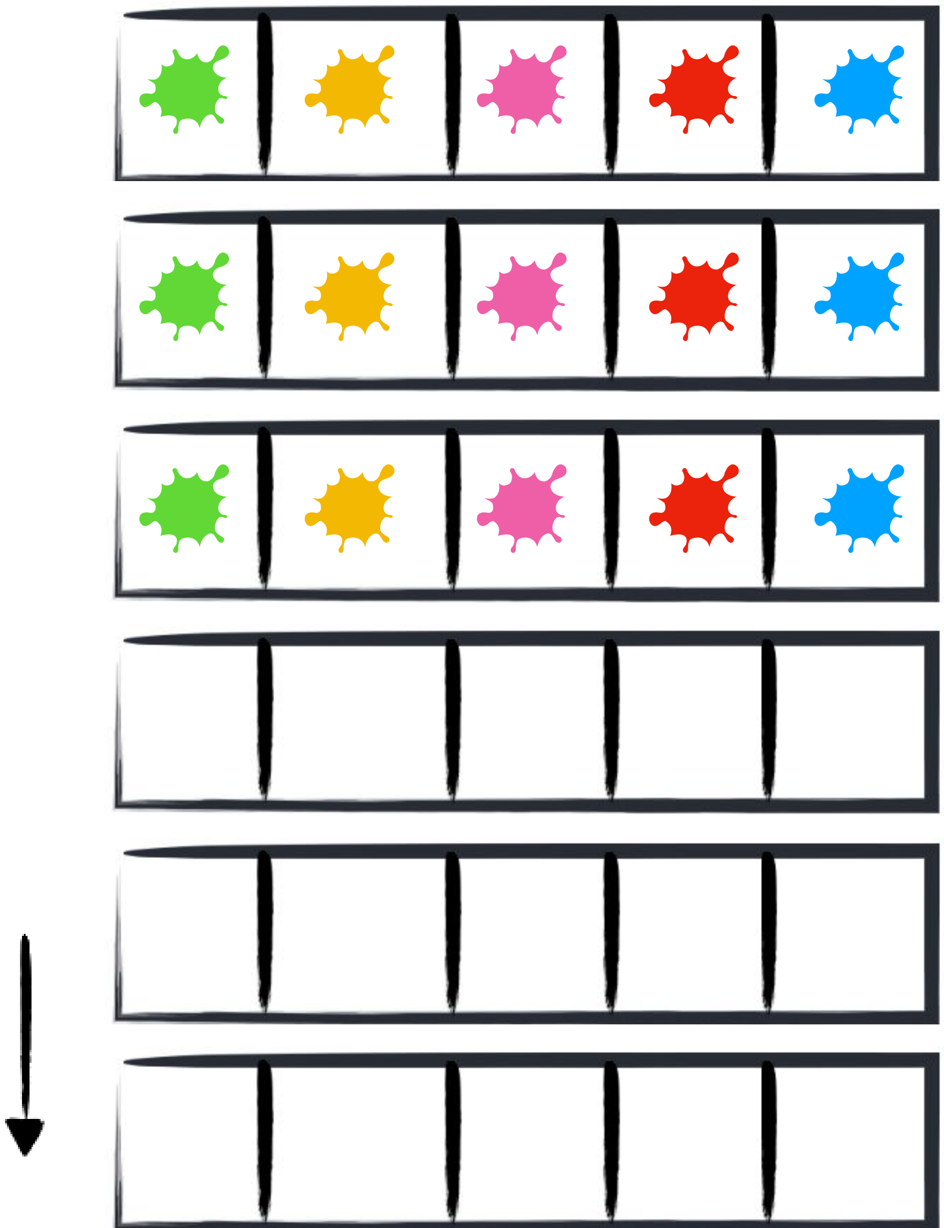


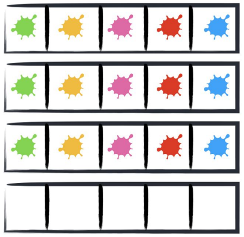




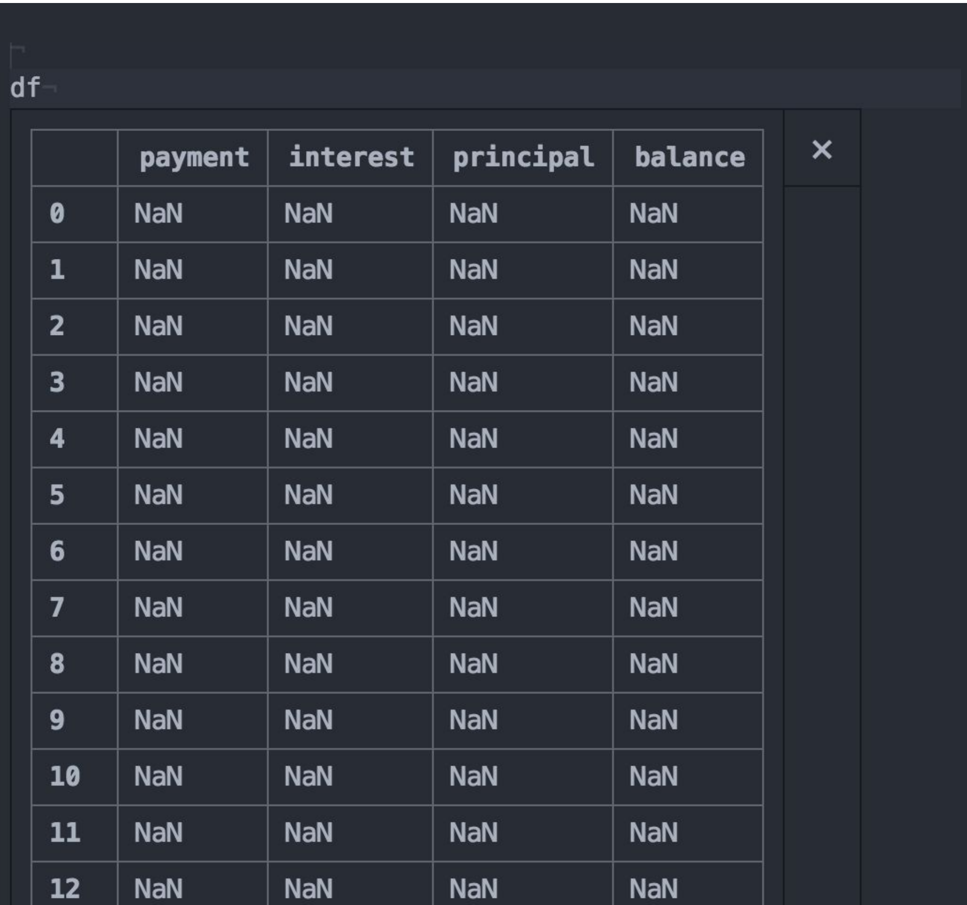






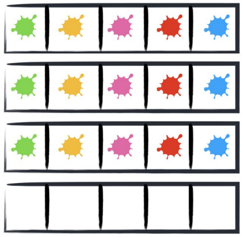


```
balance = loan
index = range(0, term)
columns = ['payment', 'interest', 'principal', 'balance']
df = pd.DataFrame(index=index, columns=columns)
```

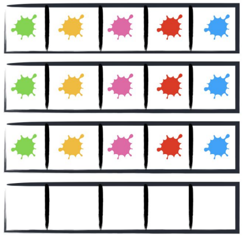


```
df
```

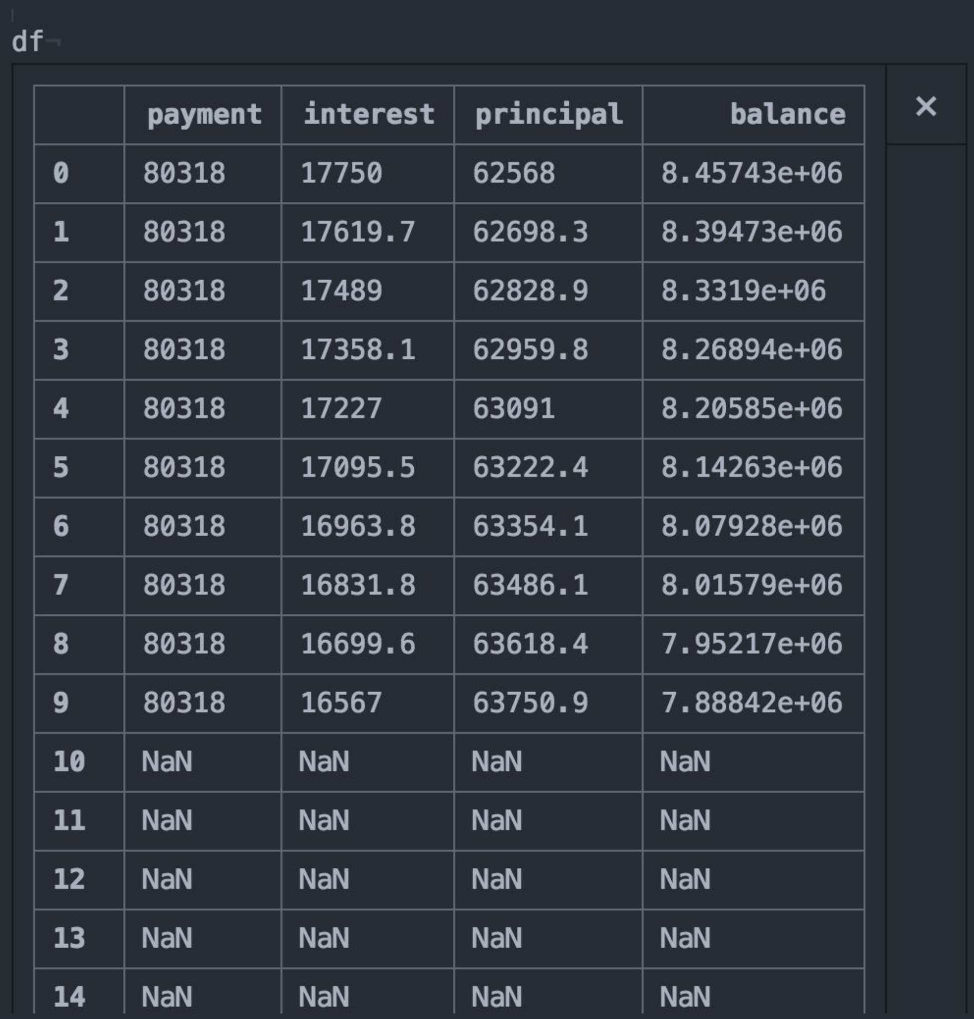
	payment	interest	principal	balance
0	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	NaN
11	NaN	NaN	NaN	NaN
12	NaN	NaN	NaN	NaN



```python
balance = loan
index = range(0, term)
columns = ['payment', 'interest', 'principal', 'balance']
df = pd.DataFrame(index=index, columns=columns)

for i in range(0, term):
    interest = round(rate/12 * balance, 2)
    principal = payment - interest
    balance = balance - principal

    df.iloc[i]['payment'] = payment
    df.iloc[i]['interest'] = interest
    df.iloc[i]['principal'] = principal
    df.iloc[i]['balance'] = balance
```

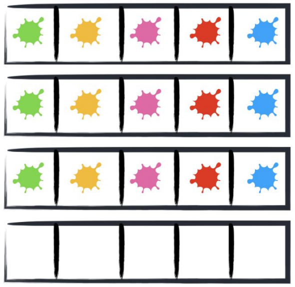


```
for i in range(0, 10):  # full term is 120
```



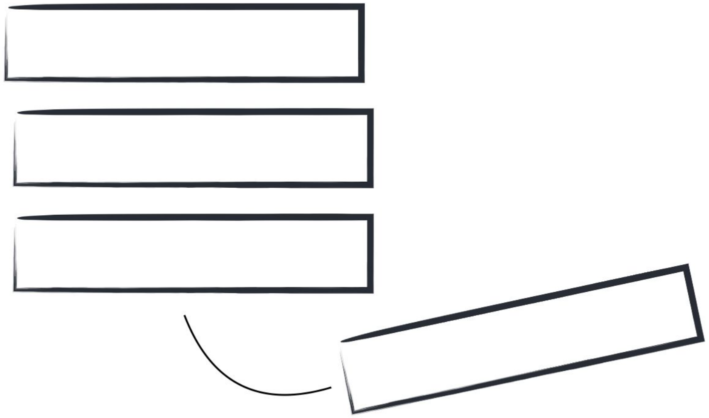
```
df
```

	payment	interest	principal	balance
0	80318	17750	62568	8.45743e+06
1	80318	17619.7	62698.3	8.39473e+06
2	80318	17489	62828.9	8.3319e+06
3	80318	17358.1	62959.8	8.26894e+06
4	80318	17227	63091	8.20585e+06
5	80318	17095.5	63222.4	8.14263e+06
6	80318	16963.8	63354.1	8.07928e+06
7	80318	16831.8	63486.1	8.01579e+06
8	80318	16699.6	63618.4	7.95217e+06
9	80318	16567	63750.9	7.88842e+06
10	NaN	NaN	NaN	NaN
11	NaN	NaN	NaN	NaN
12	NaN	NaN	NaN	NaN
13	NaN	NaN	NaN	NaN
14	NaN	NaN	NaN	NaN



```
%%timeit
good_way()
```

```
>>> 42.7 ms ± 6.38 ms per loop
>>> (mean ± std. dev. of 7 runs, 10 loops each)
```



169 ms per loop

42.7 ms per loop

```
balance = loan
index = range(0, term)
columns = ['payment', 'interest', 'principal', 'balance']
df = pd.DataFrame(index=index, columns=columns)

for i in range(0, term):
    interest = round(rate/12 * balance, 2)
    principal = payment - interest
    balance = balance - principal

    df.iloc[i]['payment'] = payment
    df.iloc[i]['interest'] = interest
    df.iloc[i]['principal'] = principal
    df.iloc[i]['balance'] = balance
```

```python
def am(loan, rate, term):

    payment = round(-np.pmt(rate/12, term, loan), 2)
    balance = loan

    index = range(0, term)
    columns = ['payment', 'interest', 'principal', 'balance']
    df = pd.DataFrame(index=index, columns=columns)

    for i in range(0, term):
        interest = round(rate/12 * balance, 2)
        principal = payment - interest
        balance = balance - principal

        df.iloc[i]['payment'] = payment
        df.iloc[i]['interest'] = interest
        df.iloc[i]['principal'] = principal
        df.iloc[i]['balance'] = balance

    return df
```

3000.00 USD

↓

8,520,000.00 COP

Banco de Bogotá

Banco AV Villas

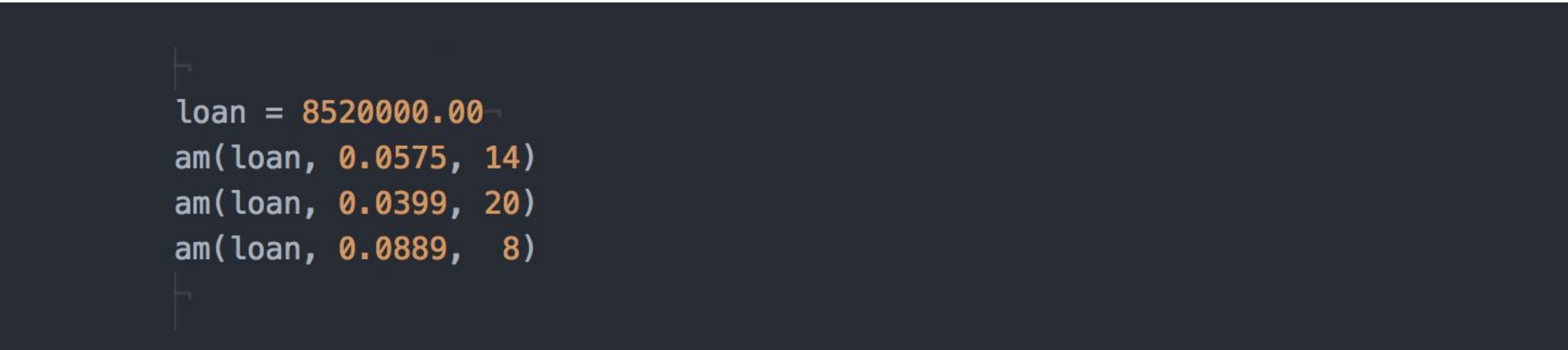
Bancolombia

8,520,000.00 COP

Banco de Bogotá	Banco AV Villas	Bancolombia
14 months	20 months	8 months
5.75%	3.99%	8.99%

8,520,000.00 COP

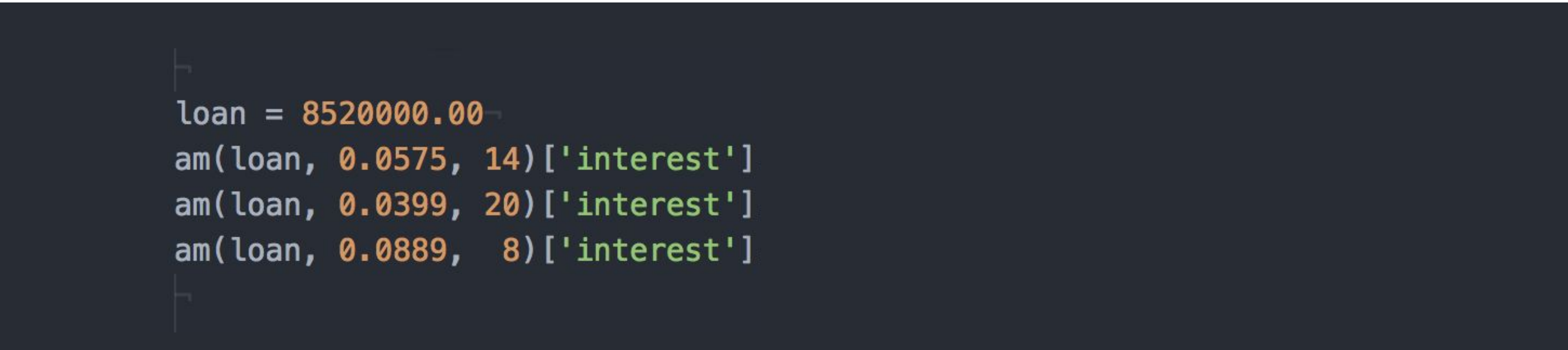
Banco de Bogotá	Banco AV Villas	Bancolombia
14 months	20 months	8 months
5.75%	3.99%	8.99%



```
loan = 8520000.00
am(loan, 0.0575, 14)
am(loan, 0.0399, 20)
am(loan, 0.0889,  8)
```

8,520,000.00 COP

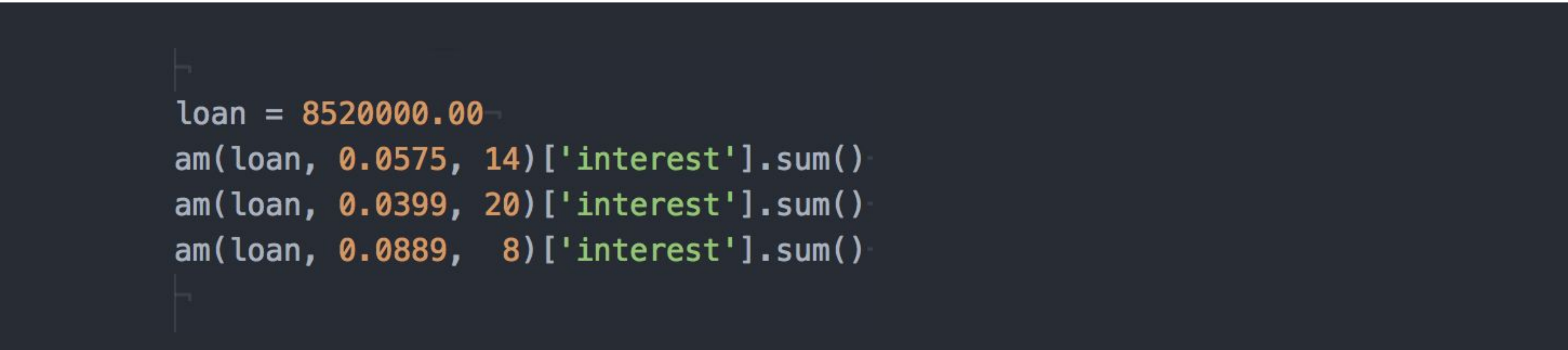
Banco de Bogotá	Banco AV Villas	Bancolombia
14 months	20 months	8 months
5.75%	3.99%	8.99%



```
loan = 8520000.00
am(loan, 0.0575, 14)['interest']
am(loan, 0.0399, 20)['interest']
am(loan, 0.0889,  8)['interest']
```

8,520,000.00 COP

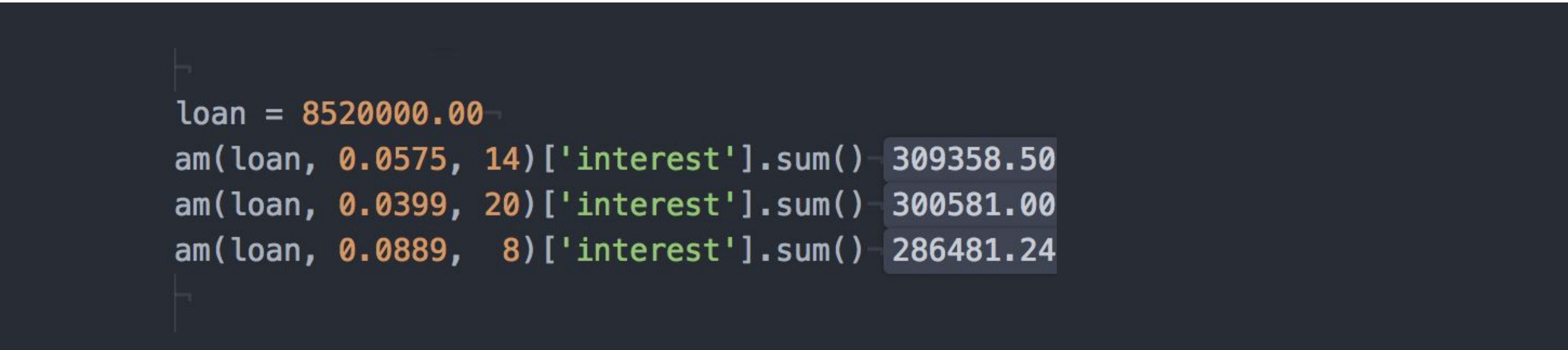
Banco de Bogotá	Banco AV Villas	Bancolombia
14 months	20 months	8 months
5.75%	3.99%	8.99%



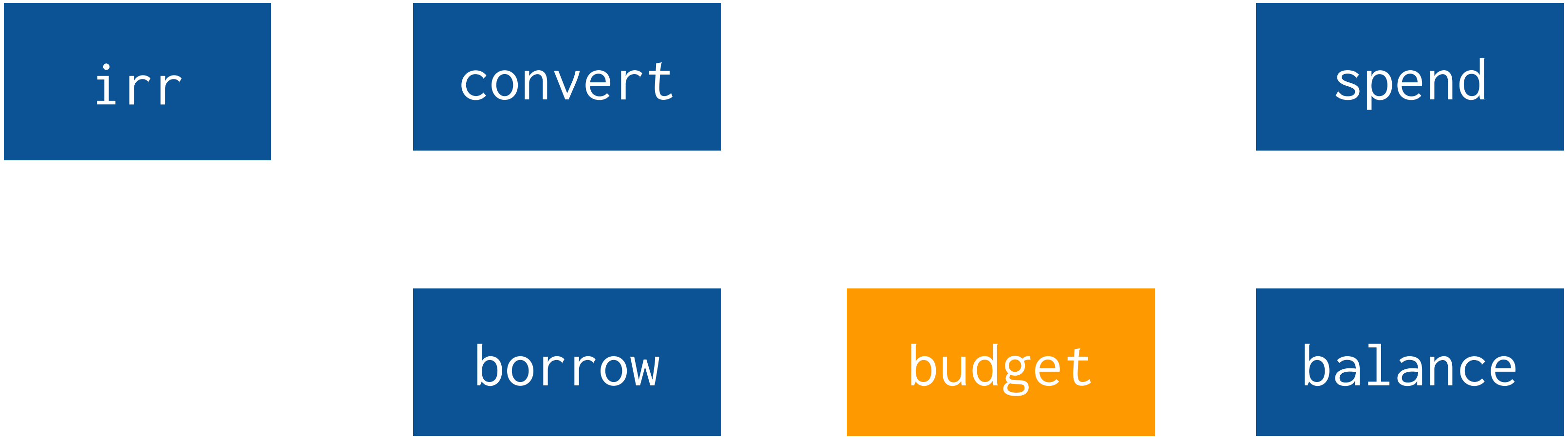
```
loan = 8520000.00
am(loan, 0.0575, 14)['interest'].sum()
am(loan, 0.0399, 20)['interest'].sum()
am(loan, 0.0889,  8)['interest'].sum()
```

8,520,000.00 COP

Banco de Bogotá	Banco AV Villas	Bancolombia
14 months	20 months	8 months
5.75%	3.99%	8.99%



```
loan = 8520000.00
am(loan, 0.0575, 14)['interest'].sum()  309358.50
am(loan, 0.0399, 20)['interest'].sum()  300581.00
am(loan, 0.0889,  8)['interest'].sum()  286481.24
```



irr

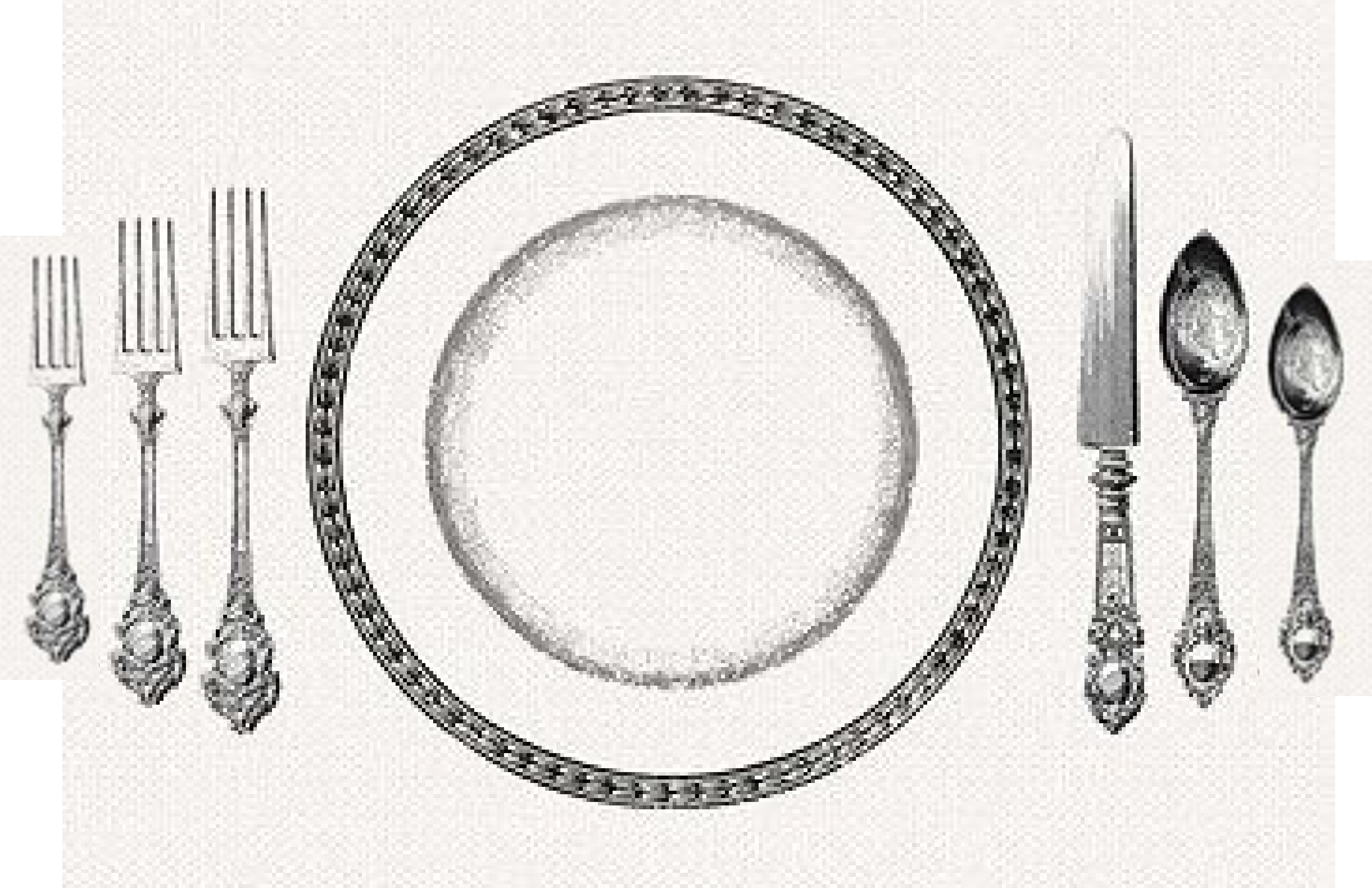
convert

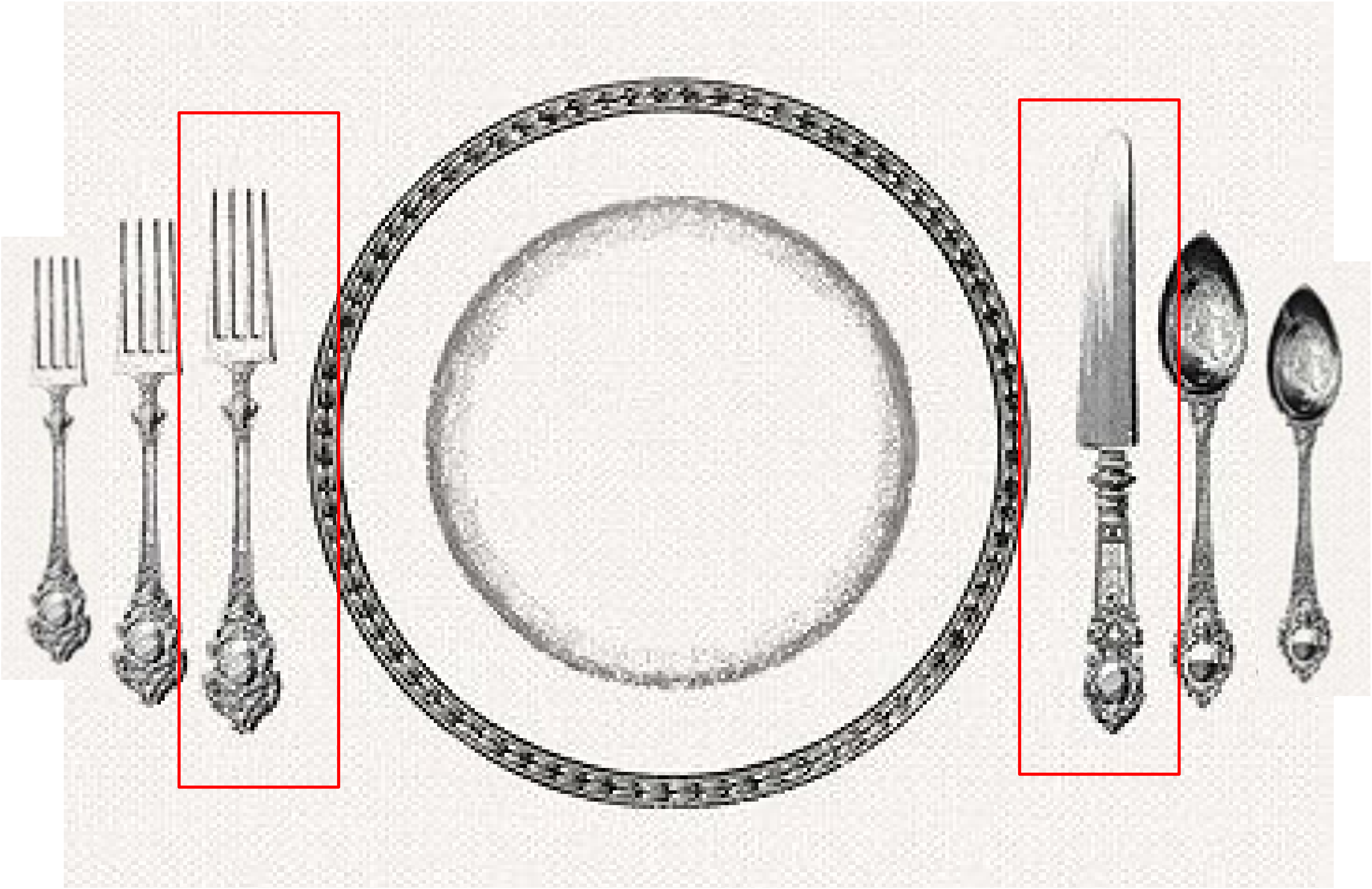
spend

borrow

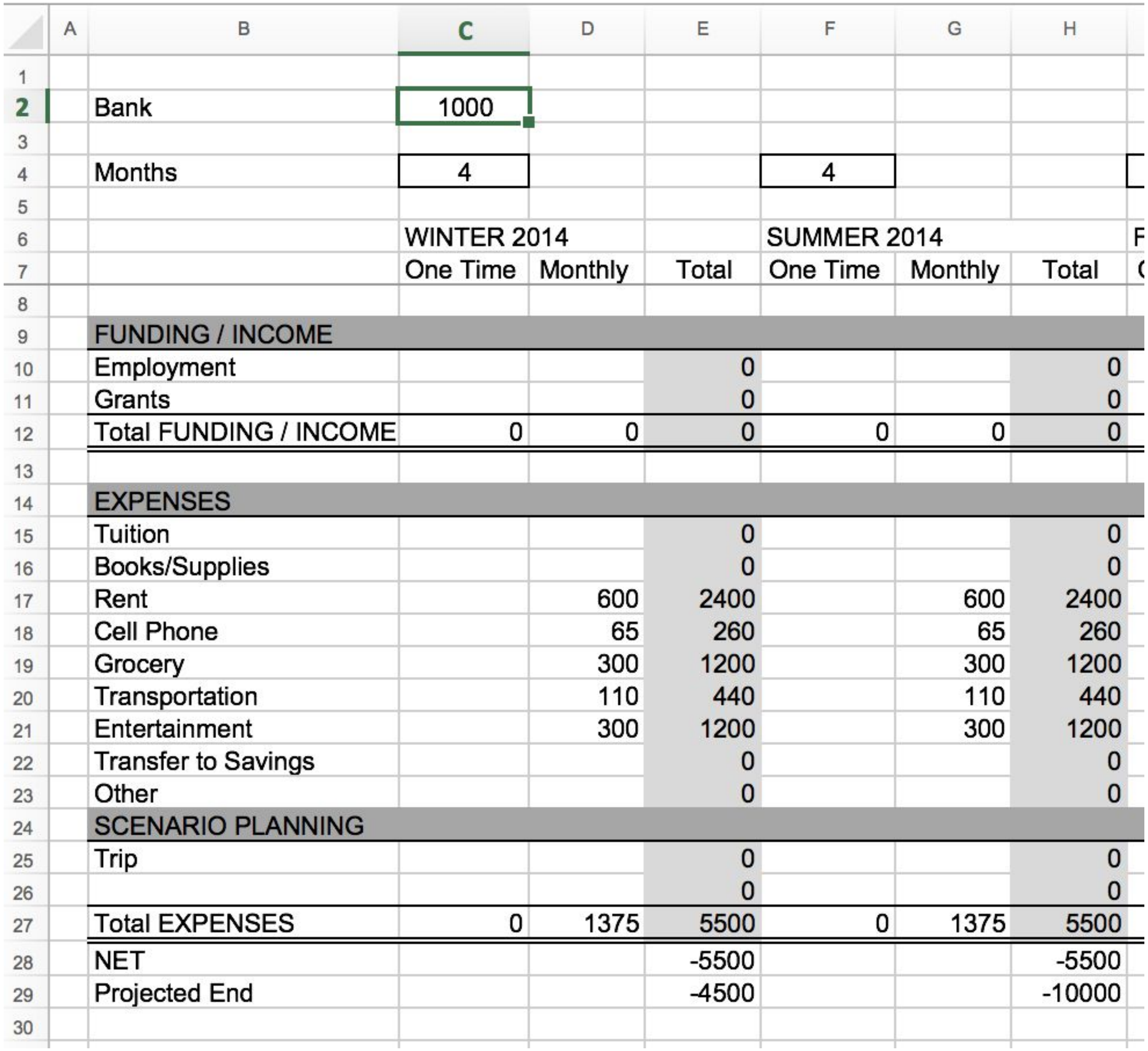
budget

balance

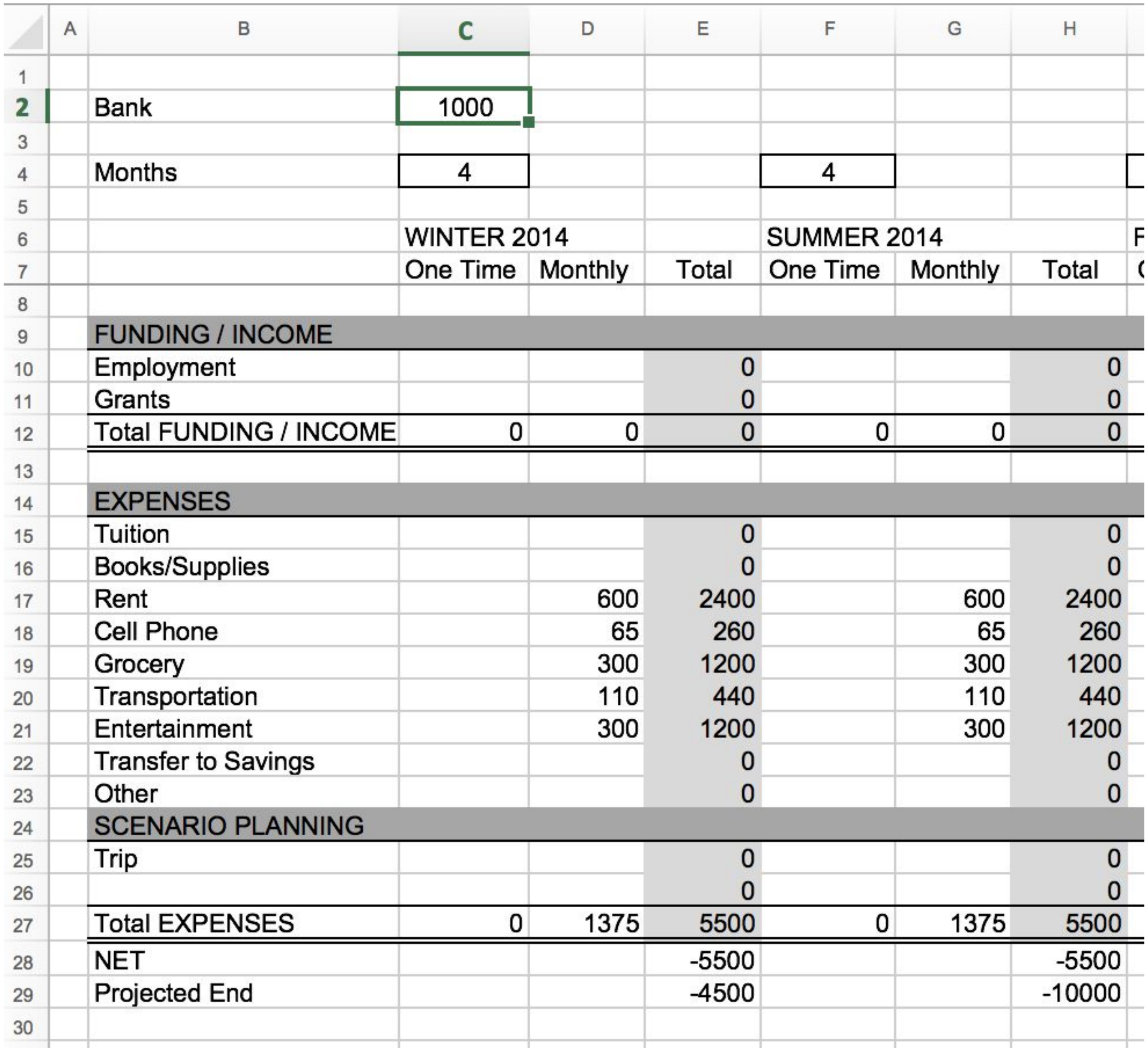




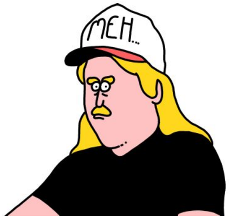
50¢
CANADIAN TIRE
CANADIAN TIRE
LA SOCIÉTÉ CANADIAN TIRE LIMITÉE
CASH BONUS • BILLET • BONI
CANADIAN TIRE CORPORATION, LIMITED
REDEEMABLE IN MERCHANDISE ONLY AT CANADIAN TIRE STORES
REMBOURSABLE EN MARCHANDISE UNIQUEMENT AUX MAGASINS CANADIAN TIRE
50
25¢
CANADIAN TIRE
CASH BONUS
CANADIAN TIRE CO

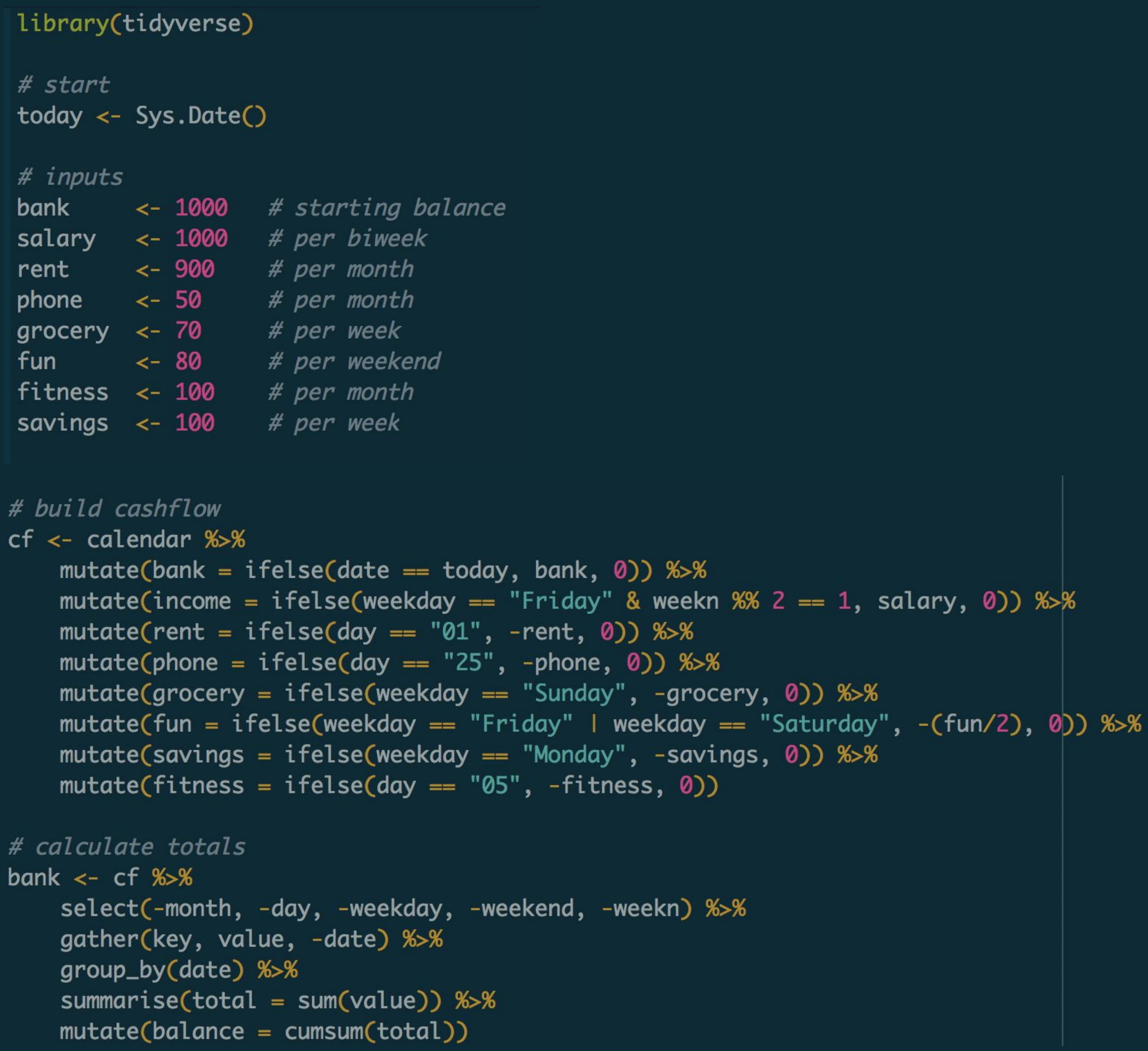


	A	B	C	D	E	F	G	H	
1									
2		Bank	1000						
3									
4		Months	4			4			
5									
6			WINTER 2014			SUMMER 2014			F
7			One Time	Monthly	Total	One Time	Monthly	Total	(
8									
9		FUNDING / INCOME							
10		Employment			0			0	
11		Grants			0			0	
12		Total FUNDING / INCOME	0	0	0	0	0	0	
13									
14		EXPENSES							
15		Tuition			0			0	
16		Books/Supplies			0			0	
17		Rent		600	2400		600	2400	
18		Cell Phone		65	260		65	260	
19		Grocery		300	1200		300	1200	
20		Transportation		110	440		110	440	
21		Entertainment		300	1200		300	1200	
22		Transfer to Savings			0			0	
23		Other			0			0	
24		SCENARIO PLANNING							
25		Trip			0			0	
26					0			0	
27		Total EXPENSES	0	1375	5500	0	1375	5500	
28		NET			-5500			-5500	
29		Projected End			-4500			-10000	
30									



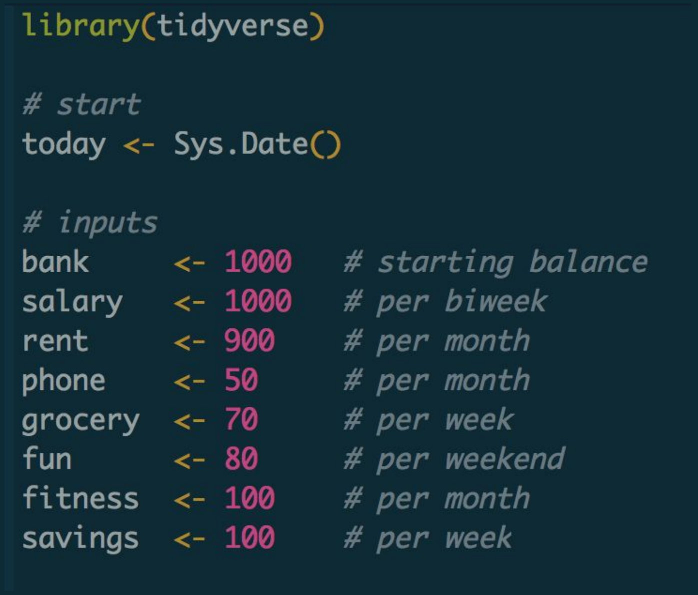
	A	B	C	D	E	F	G	H
1								
2		Bank	1000					
3								
4		Months	4			4		
5								
6			WINTER 2014			SUMMER 2014		
7			One Time	Monthly	Total	One Time	Monthly	Total
8								
9		FUNDING / INCOME						
10		Employment			0			0
11		Grants			0			0
12		Total FUNDING / INCOME	0	0	0	0	0	0
13								
14		EXPENSES						
15		Tuition			0			0
16		Books/Supplies			0			0
17		Rent		600	2400		600	2400
18		Cell Phone		65	260		65	260
19		Grocery		300	1200		300	1200
20		Transportation		110	440		110	440
21		Entertainment		300	1200		300	1200
22		Transfer to Savings			0			0
23		Other			0			0
24		SCENARIO PLANNING						
25		Trip			0			0
26					0			0
27		Total EXPENSES	0	1375	5500	0	1375	5500
28		NET			-5500			-5500
29		Projected End			-4500			-10000
30								



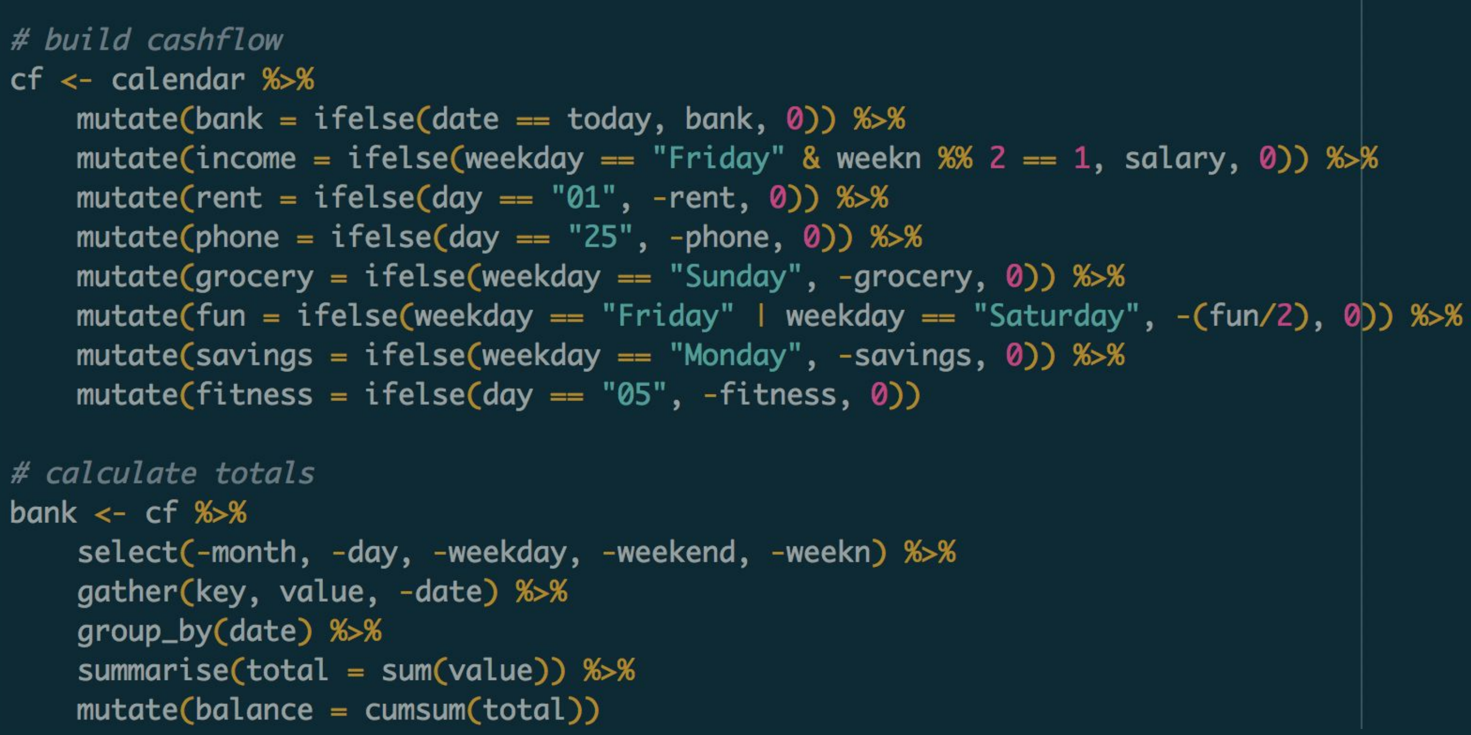


```r
library(tidyverse)

# start
today <- Sys.Date()

# inputs
bank     <- 1000   # starting balance
salary   <- 1000   # per biweek
rent     <- 900    # per month
phone    <- 50     # per month
grocery  <- 70     # per week
fun      <- 80     # per weekend
fitness  <- 100    # per month
savings  <- 100    # per week
```

```r
# build cashflow
cf <- calendar %>%
    mutate(bank = ifelse(date == today, bank, 0)) %>%
    mutate(income = ifelse(weekday == "Friday" & weekn %% 2 == 1, salary, 0)) %>%
    mutate(rent = ifelse(day == "01", -rent, 0)) %>%
    mutate(phone = ifelse(day == "25", -phone, 0)) %>%
    mutate(grocery = ifelse(weekday == "Sunday", -grocery, 0)) %>%
    mutate(fun = ifelse(weekday == "Friday" | weekday == "Saturday", -(fun/2), 0)) %>%
    mutate(savings = ifelse(weekday == "Monday", -savings, 0)) %>%
    mutate(fitness = ifelse(day == "05", -fitness, 0))

# calculate totals
bank <- cf %>%
    select(-month, -day, -weekday, -weekend, -weekn) %>%
    gather(key, value, -date) %>%
    group_by(date) %>%
    summarise(total = sum(value)) %>%
    mutate(balance = cumsum(total))
```



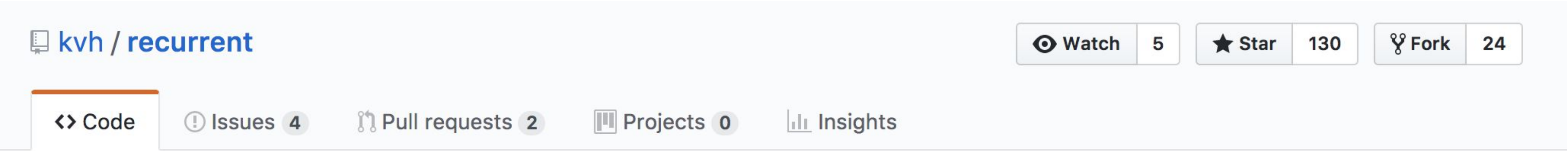
```r
library(tidyverse)

# start
today <- Sys.Date()

# inputs
bank     <- 1000   # starting balance
salary   <- 1000   # per biweek
rent     <- 900    # per month
phone    <- 50     # per month
grocery  <- 70     # per week
fun      <- 80     # per weekend
fitness  <- 100    # per month
savings  <- 100    # per week
```



```r
# build cashflow
cf <- calendar %>%
    mutate(bank = ifelse(date == today, bank, 0)) %>%
    mutate(income = ifelse(weekday == "Friday" & weekn %% 2 == 1, salary, 0)) %>%
    mutate(rent = ifelse(day == "01", -rent, 0)) %>%
    mutate(phone = ifelse(day == "25", -phone, 0)) %>%
    mutate(grocery = ifelse(weekday == "Sunday", -grocery, 0)) %>%
    mutate(fun = ifelse(weekday == "Friday" | weekday == "Saturday", -(fun/2), 0)) %>%
    mutate(savings = ifelse(weekday == "Monday", -savings, 0)) %>%
    mutate(fitness = ifelse(day == "05", -fitness, 0))

# calculate totals
bank <- cf %>%
    select(-month, -day, -weekday, -weekend, -weekn) %>%
    gather(key, value, -date) %>%
    group_by(date) %>%
    summarise(total = sum(value)) %>%
    mutate(balance = cumsum(total))
```

$200	every other day
$32	every 16th of the month
$567	first thursday of every month
$100	third and fourth friday of each month
$56	weekly on wednesdays and fridays
$2	every day starting next tuesday until feb
$600	every week on sunday starting tomorrow until November
$1000	tomorrow

```
every other day
every 16th of the month
first thursday of every month
third and fourth friday of each month
weekly on wednesdays and fridays
every day starting next tuesday until feb
every week on sunday starting tomorrow until November
tomorrow
```



kvh / recurrent

```
r = RecurringEvent()
r.parse('every other day')
r.parse('every 16th of the month')
r.parse('first thursday of every month')
r.parse('third and fourth friday of each month')
r.parse('weekly on wednesdays and fridays')
r.parse('every day starting next tuesday until feb')
r.parse('every week on sunday starting tomorrow until November')
r.parse('tomorrow')
```

```
import datetime
from dateutil import rrule
from recurrent import RecurringEvent
```

```
r = RecurringEvent()
r.parse('every other day')
r.parse('every 16th of the month')
r.parse('first thursday of every month')
r.parse('third and fourth friday of each month')
r.parse('weekly on wednesdays and fridays')
r.parse('every day starting next tuesday until feb')
r.parse('every week on sunday starting tomorrow until November')
r.parse('tomorrow')
```

```
'RRULE:INTERVAL=2;FREQ=DAILY'
r.parse('every 16th of the month')
r.parse('first thursday of every month')
r.parse('third and fourth friday of each month')
r.parse('weekly on wednesdays and fridays')
r.parse('every day starting next tuesday until feb')
r.parse('every week on sunday starting tomorrow until November')
r.parse('tomorrow')
```

```
'RRULE:INTERVAL=2;FREQ=DAILY'
'RRULE:BYMONTHDAY=16;INTERVAL=1;FREQ=MONTHLY'
r.parse('first thursday of every month')
r.parse('third and fourth friday of each month')
r.parse('weekly on wednesdays and fridays')
r.parse('every day starting next tuesday until feb')
r.parse('every week on sunday starting tomorrow until November')
r.parse('tomorrow')
```

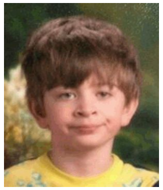
```
'RRULE:INTERVAL=2;FREQ=DAILY'
'RRULE:BYMONTHDAY=16;INTERVAL=1;FREQ=MONTHLY'
'RRULE:BYDAY=1TH;INTERVAL=1;FREQ=MONTHLY'
r.parse('third and fourth friday of each month')
r.parse('weekly on wednesdays and fridays')
r.parse('every day starting next tuesday until feb')
r.parse('every week on sunday starting tomorrow until November')
r.parse('tomorrow')
```

```
'RRULE:INTERVAL=2;FREQ=DAILY'
'RRULE:BYMONTHDAY=16;INTERVAL=1;FREQ=MONTHLY'
'RRULE:BYDAY=1TH;INTERVAL=1;FREQ=MONTHLY'
'RRULE:BYDAY=3FR,4FR;INTERVAL=1;FREQ=MONTHLY'
r.parse('weekly on wednesdays and fridays')
r.parse('every day starting next tuesday until feb')
r.parse('every week on sunday starting tomorrow until November')
r.parse('tomorrow')
```

```
'RRULE:INTERVAL=2;FREQ=DAILY'
'RRULE:BYMONTHDAY=16;INTERVAL=1;FREQ=MONTHLY'
'RRULE:BYDAY=1TH;INTERVAL=1;FREQ=MONTHLY'
'RRULE:BYDAY=3FR,4FR;INTERVAL=1;FREQ=MONTHLY'
'RRULE:BYDAY=WE,FR;INTERVAL=1;FREQ=WEEKLY'
'DTSTART:20180213\nRRULE:INTERVAL=1;FREQ=DAILY;UNTIL=20190201'
'DTSTART:20180206\nRRULE:BYDAY=SU;INTERVAL=1;FREQ=WEEKLY;UNTIL=20181101'
r.parse('tomorrow')
```

```
'RRULE:INTERVAL=2;FREQ=DAILY'
'RRULE:BYMONTHDAY=16;INTERVAL=1;FREQ=MONTHLY'
'RRULE:BYDAY=1TH;INTERVAL=1;FREQ=MONTHLY'
'RRULE:BYDAY=3FR,4FR;INTERVAL=1;FREQ=MONTHLY'
'RRULE:BYDAY=WE,FR;INTERVAL=1;FREQ=WEEKLY'
'DTSTART:20180213\nRRULE:INTERVAL=1;FREQ=DAILY;UNTIL=20190201'
'DTSTART:20180206\nRRULE:BYDAY=SU;INTERVAL=1;FREQ=WEEKLY;UNTIL=20181101'
datetime.datetime(2018, 2, 6, 9, 0)
```

```
'RRULE:INTERVAL=2;FREQ=DAILY'
'RRULE:BYMONTHDAY=16;INTERVAL=1;FREQ=MONTHLY'
'RRULE:BYDAY=1TH;INTERVAL=1;FREQ=MONTHLY'
'RRULE:BYDAY=3FR,4FR;INTERVAL=1;FREQ=MONTHLY'
'RRULE:BYDAY=WE,FR;INTERVAL=1;FREQ=WEEKLY'
'DTSTART:20180213\nRRULE:INTERVAL=1;FREQ=DAILY;UNTIL=20190201'
'DTSTART:20180206\nRRULE:BYDAY=SU;INTERVAL=1;FREQ=WEEKLY;UNTIL=20181101'
datetime.datetime(2018, 2, 6, 9, 0)
```



```
r = RecurringEvent()
r.parse('every 3 weeks starting 2018-05-01 until 2018-09-30')
```

```
r = RecurringEvent()
r.parse('every 3 weeks starting 2018-05-01 until 2018-09-30')
```

```
>>> ‘DTSTART:20180501\nRRULE:INTERVAL=3;FREQ=WEEKLY;UNTIL=20180930'
```

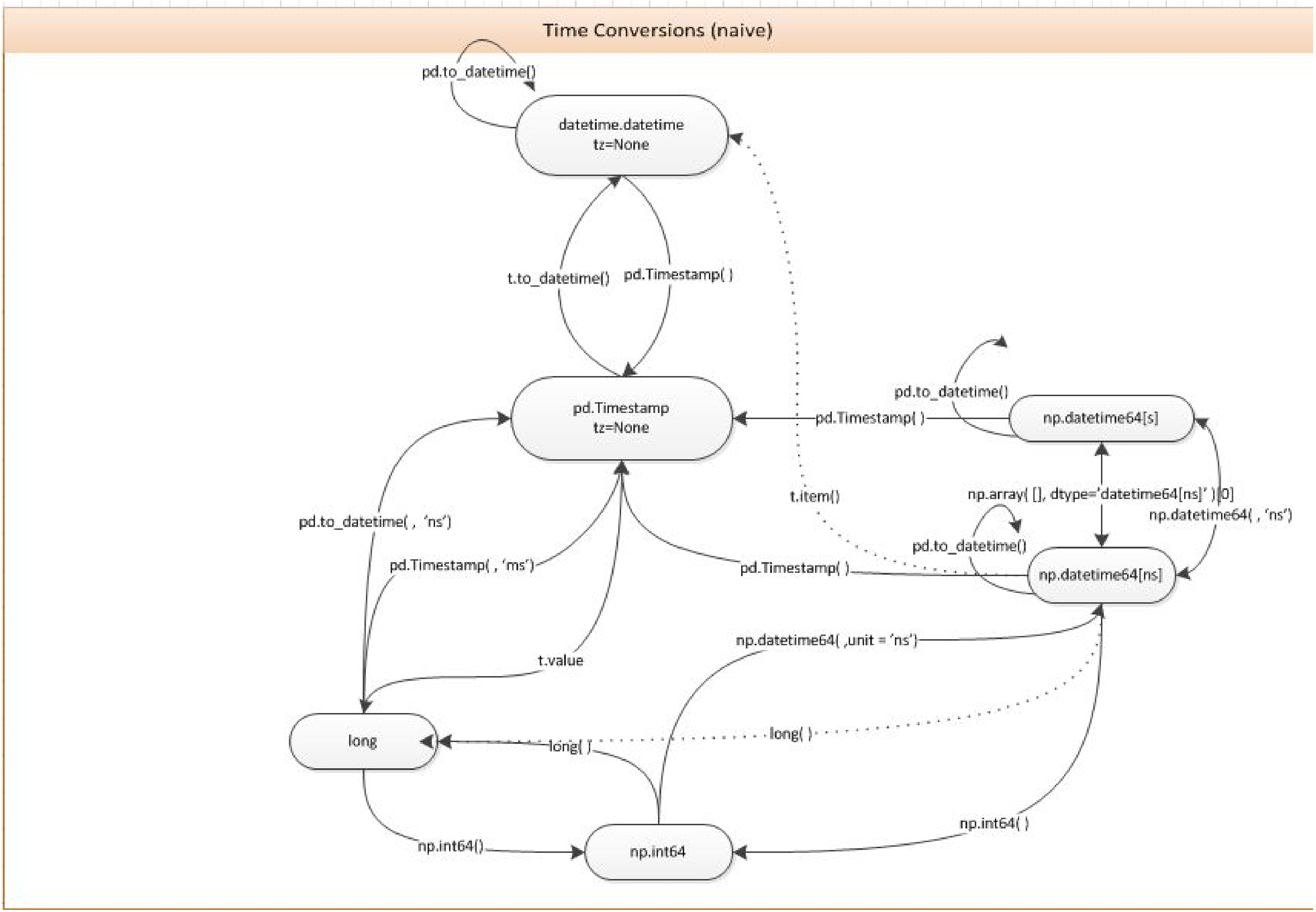
```
r = RecurringEvent()
r.parse('every 3 weeks starting 2018-05-01 until 2018-09-30')
>>> ‘DTSTART:20180501\nRRULE:INTERVAL=3;FREQ=WEEKLY;UNTIL=20180930'
```

```
rr = rrule.rrulestr(r.get_RFC_rrule())
rr.after(datetime.datetime.now())
>>> datetime.datetime(2018, 5, 1, 0, 0)
rr.count()
>>> 8
rr.before(datetime.datetime(2018, 7, 1))
>>> datetime.datetime(2018, 6, 12, 0, 0)
```

```
r = RecurringEvent()
r.parse('every 3 weeks starting 2018-05-01 until 2018-09-30')
rr = rrule.rrulestr(r.get_RFC_rrule())
rr.between(datetime.date.today(), datetime.date(2018, 9, 1))
```

```python
r = RecurringEvent()
r.parse('every 3 weeks starting 2018-05-01 until 2018-09-30')
rr = rrule.rrulestr(r.get_RFC_rrule())
rr.between(datetime.date.today(), datetime.date(2018, 9, 1))
```

```
-----------------------------------------------------------------
TypeError                       Traceback (most recent call last)
<ipython-input-46-77d7d68d1920> in <module>()
--> 1 rr.between(datetime.date.today(), datetime.date(2018, 9, 1))

TypeError: can't compare datetime.datetime to datetime.date
```



https://stackoverflow.com/questions/13703720/converting-between-datetime-timestamp-and-datetime64/21916253#21916253

```
r = RecurringEvent()
r.parse('every 3 weeks starting 2018-05-01 until 2018-09-30')
rr = rrule.rrulestr(r.get_RFC_rrule())
rr.between(datetime.date.today(), datetime.date(2018, 9, 1))
```

```
r = RecurringEvent()
r.parse('every 3 weeks starting 2018-05-01 until 2018-09-30')
rr = rrule.rrulestr(r.get_RFC_rrule())
rr.between(datetime.datetime.now(), datetime.datetime(2018, 9, 1))
```

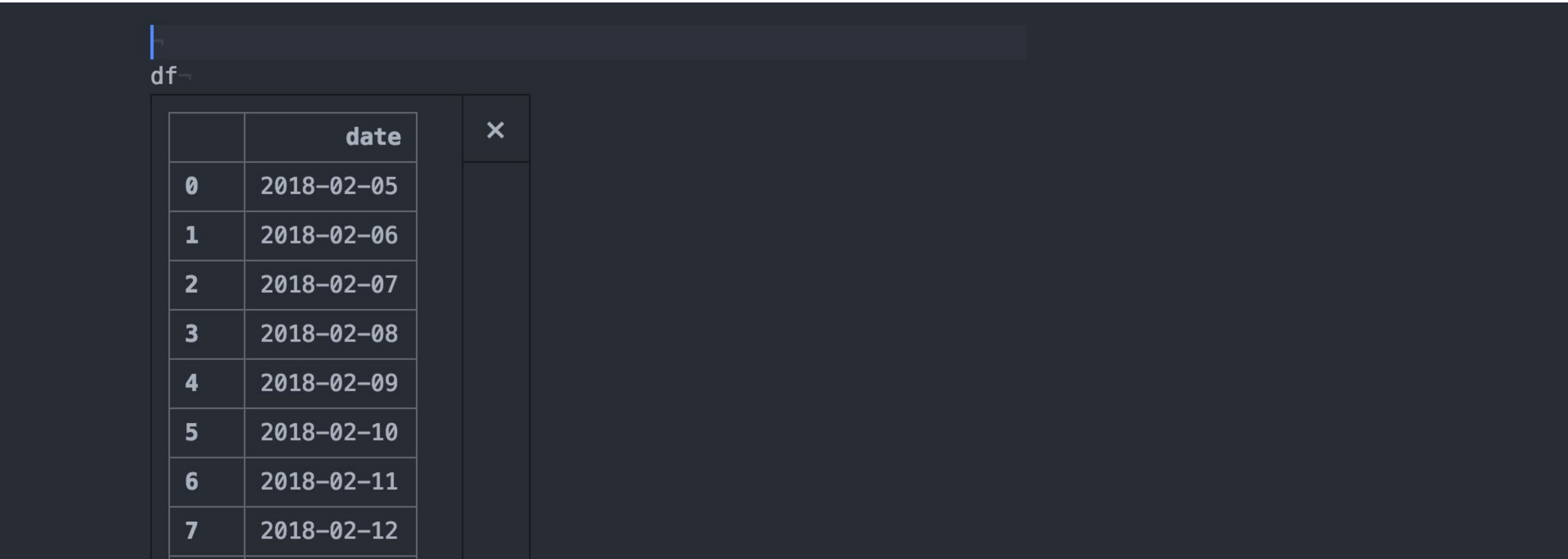
```
r = RecurringEvent()
r.parse('every 3 weeks starting 2018-05-01 until 2018-09-30')
rr = rrule.rrulestr(r.get_RFC_rrule())
rr.between(datetime.datetime.now(), datetime.datetime(2018, 9, 1))
```

```
[datetime.datetime(2018, 5, 1, 0, 0),
 datetime.datetime(2018, 5, 22, 0, 0),
 datetime.datetime(2018, 6, 12, 0, 0),
 datetime.datetime(2018, 7, 3, 0, 0),
 datetime.datetime(2018, 7, 24, 0, 0),
 datetime.datetime(2018, 8, 14, 0, 0)]
```

```python
r = RecurringEvent()
r.parse('every 3 weeks starting 2018-05-01 until 2018-09-30')
rr = rrule.rrulestr(r.get_RFC_rrule())
rr.between(datetime.datetime.now(), datetime.datetime(2018, 9, 1))
```

```
[datetime.datetime(2018, 5, 1, 0, 0),
 datetime.datetime(2018, 5, 22, 0, 0),
 datetime.datetime(2018, 6, 12, 0, 0),
 datetime.datetime(2018, 7, 3, 0, 0),
 datetime.datetime(2018, 7, 24, 0, 0),
 datetime.datetime(2018, 8, 14, 0, 0)]
```

```python
TODAY = normalize_datetime(datetime.datetime.now())
END = TODAY + datetime.timedelta(days=365)

df = pd.DataFrame({
    'date': pd.date_range(
        start=TODAY,
        end=END,
        normalize=True,
        freq='D')
})
```

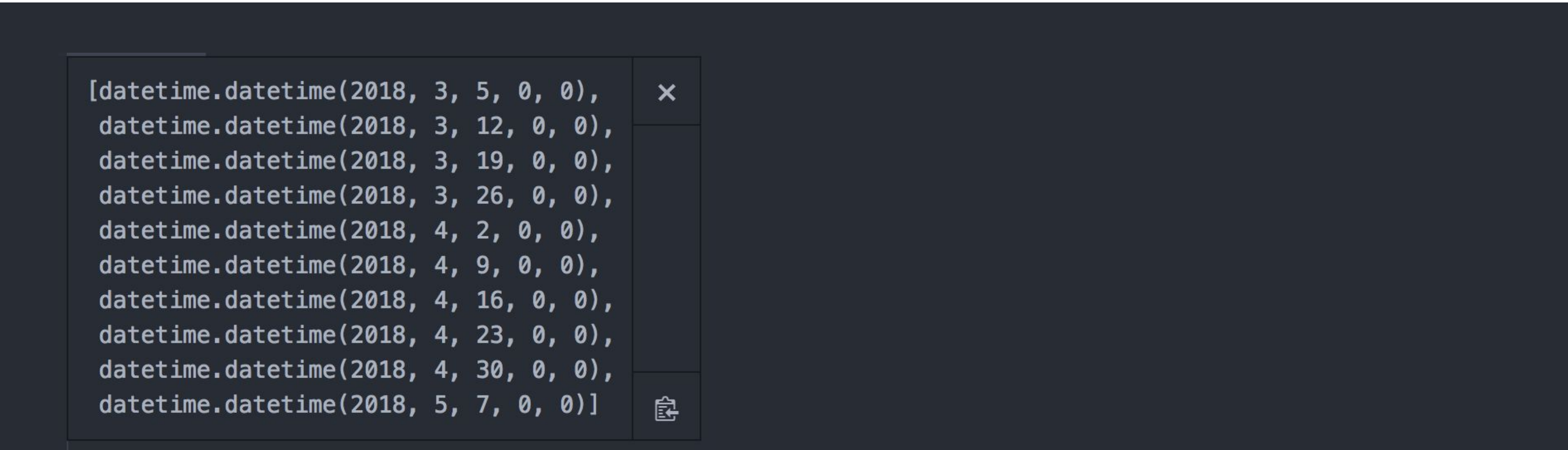


```python
df
```

	date
0	2018-02-05
1	2018-02-06
2	2018-02-07
3	2018-02-08
4	2018-02-09
5	2018-02-10
6	2018-02-11
7	2018-02-12

```
things = {
    'mining_income': {
        'amount': 100,
        'frequency': 'every monday starting in March'
    }
}
```

```
things = {
    'mining_income': {
        'amount': 100,
        'frequency': 'every monday starting in March'
    }
}
amount = things['mining_income']['amount']
rr = get_rrule_or_datetime(things['mining_income']['frequency'])
dates = rr.between(TODAY, END)
dates = [normalize_datetime(d) for d in dates]
dates[:10]
```



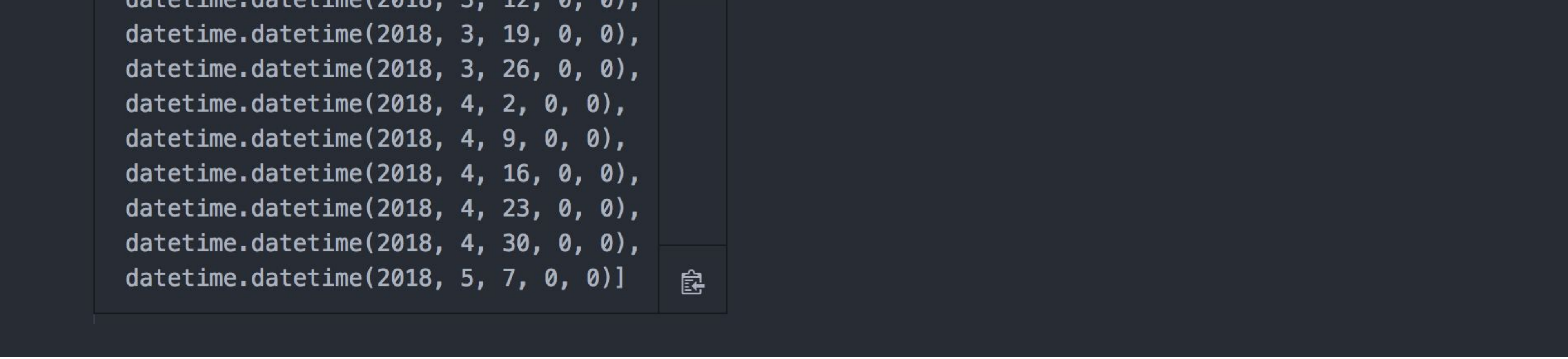
```
[datetime.datetime(2018, 3, 5, 0, 0),
 datetime.datetime(2018, 3, 12, 0, 0),
 datetime.datetime(2018, 3, 19, 0, 0),
 datetime.datetime(2018, 3, 26, 0, 0),
 datetime.datetime(2018, 4, 2, 0, 0),
 datetime.datetime(2018, 4, 9, 0, 0),
 datetime.datetime(2018, 4, 16, 0, 0),
 datetime.datetime(2018, 4, 23, 0, 0),
 datetime.datetime(2018, 4, 30, 0, 0),
 datetime.datetime(2018, 5, 7, 0, 0)]
```

```
def get_rrule_or_datetime(frequency):
    try:
        r = RecurringEvent()
        f = r.parse(frequency)
        return rrule.rrulestr(r.get_RFC_rrule())
    except ValueError:  # r.parse() returned a datetime.datetime
        return f
    except AttributeError:  # frequency is a datetime.date
        return datetime.datetime.combine(frequency, datetime.time())


def normalize_datetime(dt):
    return datetime.datetime.combine(dt, datetime.time())
```

```
import this

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
```

```
import this

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
```

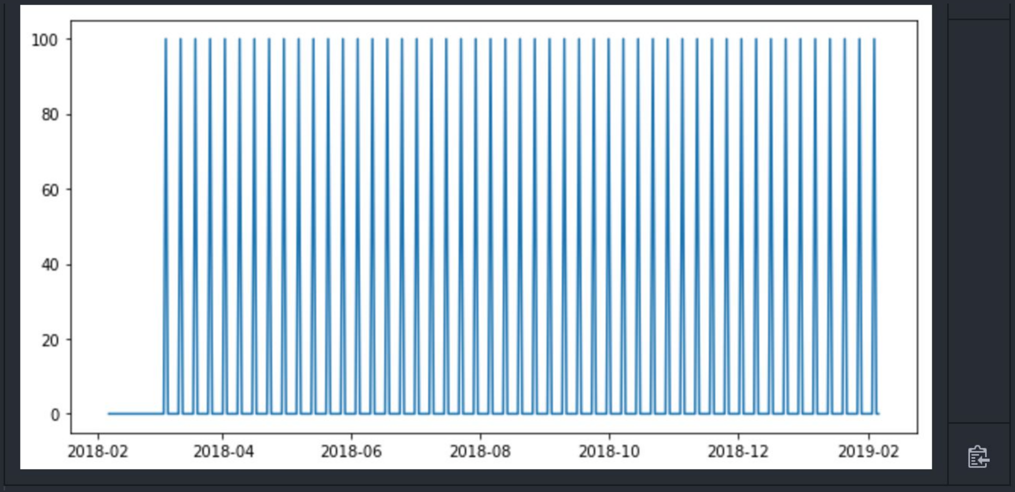
```
import this

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
```

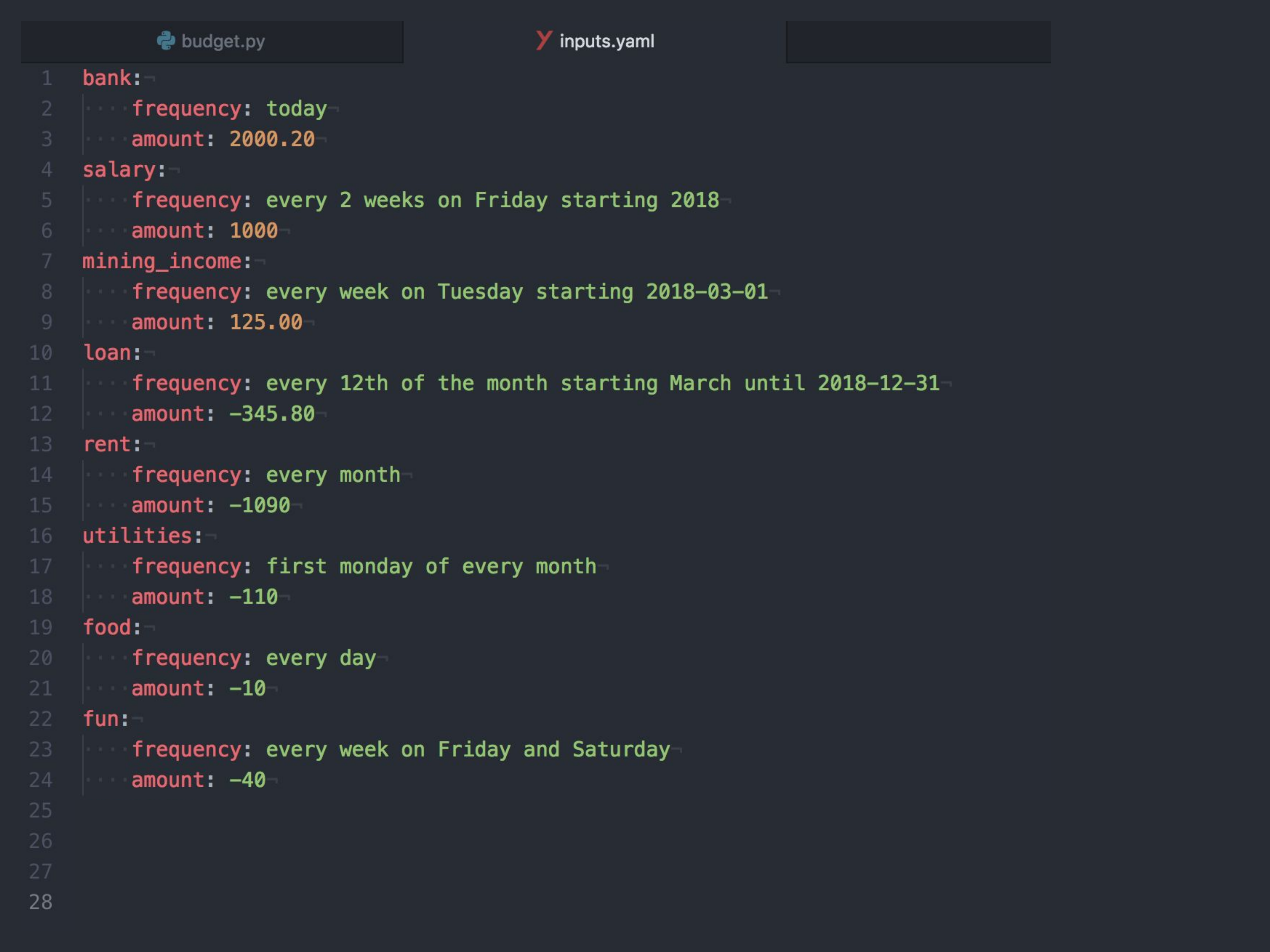


```
datetime.datetime(2018, 3, 12, 0, 0),
datetime.datetime(2018, 3, 19, 0, 0),
datetime.datetime(2018, 3, 26, 0, 0),
datetime.datetime(2018, 4, 2, 0, 0),
datetime.datetime(2018, 4, 9, 0, 0),
datetime.datetime(2018, 4, 16, 0, 0),
datetime.datetime(2018, 4, 23, 0, 0),
datetime.datetime(2018, 4, 30, 0, 0),
datetime.datetime(2018, 5, 7, 0, 0)]
```

```
df = df.merge(
    pd.DataFrame({'date': dates, 'mining_income': amount}),
    how='left').fillna(0)

plt.figure(figsize=(10, 5))
plt.plot(df.date, df.mining_income)
```

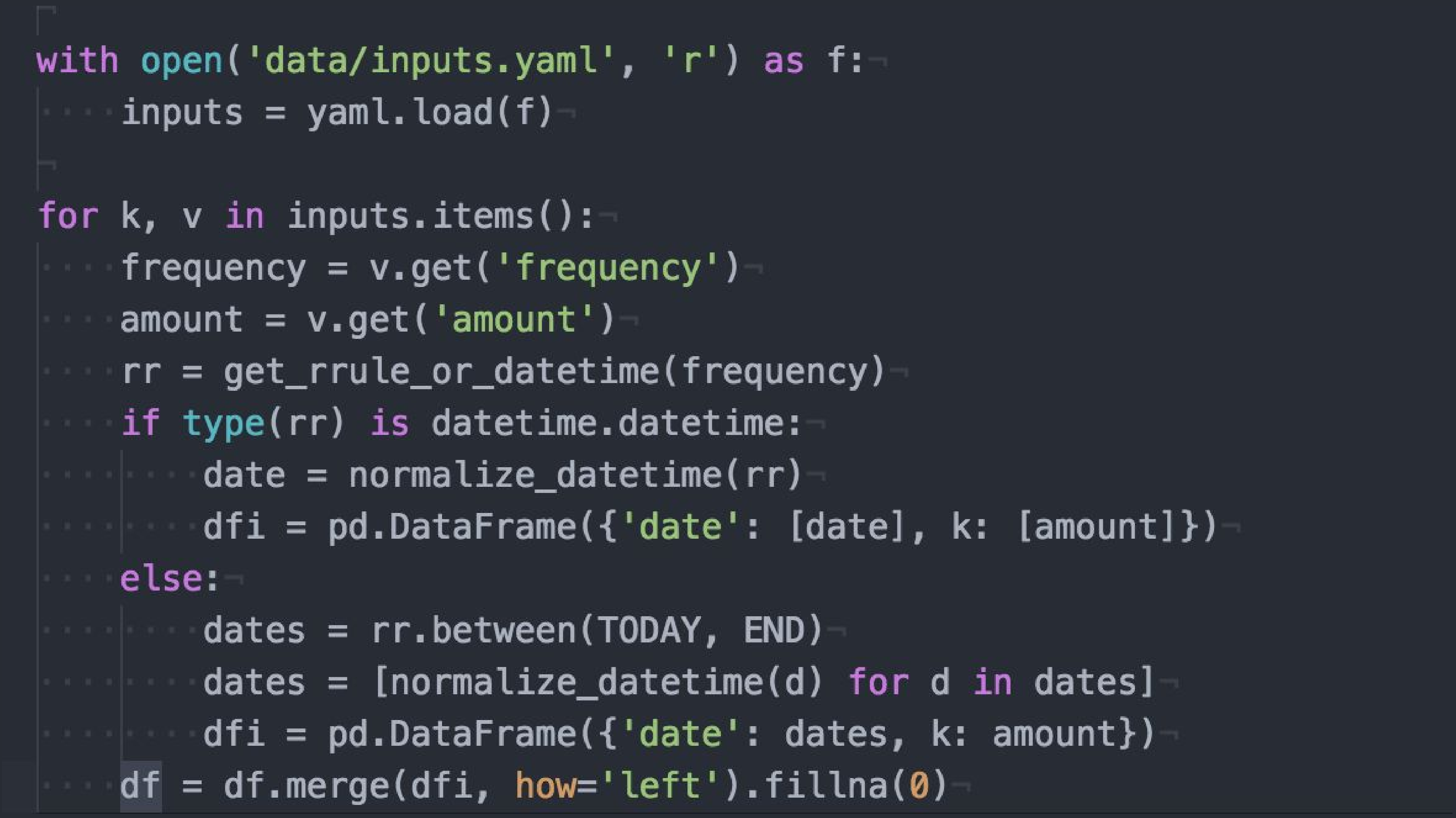
```
df = df.merge(
    pd.DataFrame({'date': dates, 'mining_income': amount}),
    how='left').fillna(0)

plt.figure(figsize=(10, 5))
plt.plot(df.date, df.mining_income)
```



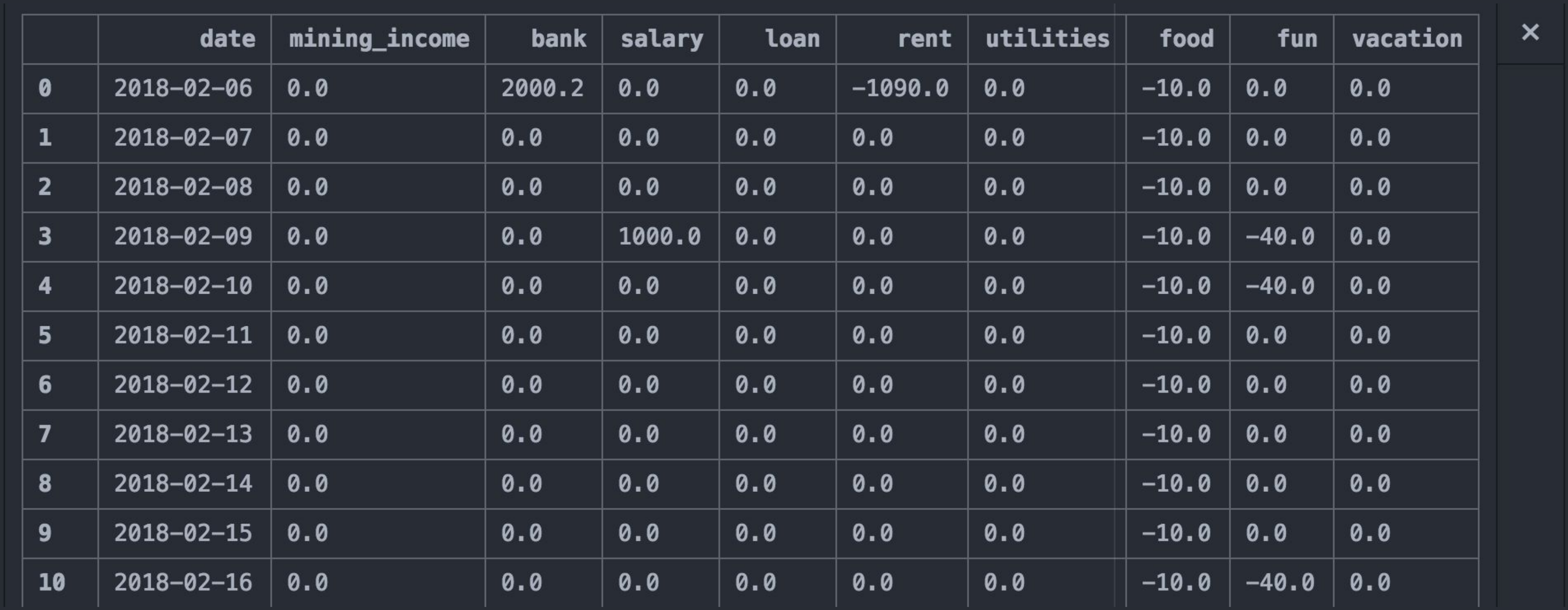


```yaml
bank:
    frequency: today
    amount: 2000.20
salary:
    frequency: every 2 weeks on Friday starting 2018
    amount: 1000
mining_income:
    frequency: every week on Tuesday starting 2018-03-01
    amount: 125.00
loan:
    frequency: every 12th of the month starting March until 2018-12-31
    amount: -345.80
rent:
    frequency: every month
    amount: -1090
utilities:
    frequency: first monday of every month
    amount: -110
food:
    frequency: every day
    amount: -10
fun:
    frequency: every week on Friday and Saturday
    amount: -40
```

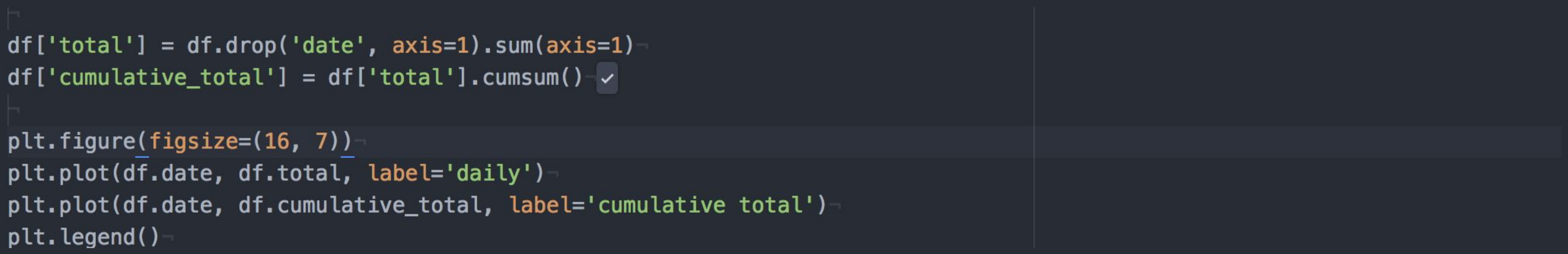
```
TODAY = normalize_datetime(datetime.datetime.now())
END = TODAY + datetime.timedelta(days=365)

df = pd.DataFrame({
    'date': pd.date_range(
        start=TODAY,
        end=END,
        normalize=True,
        freq='D')
})
```



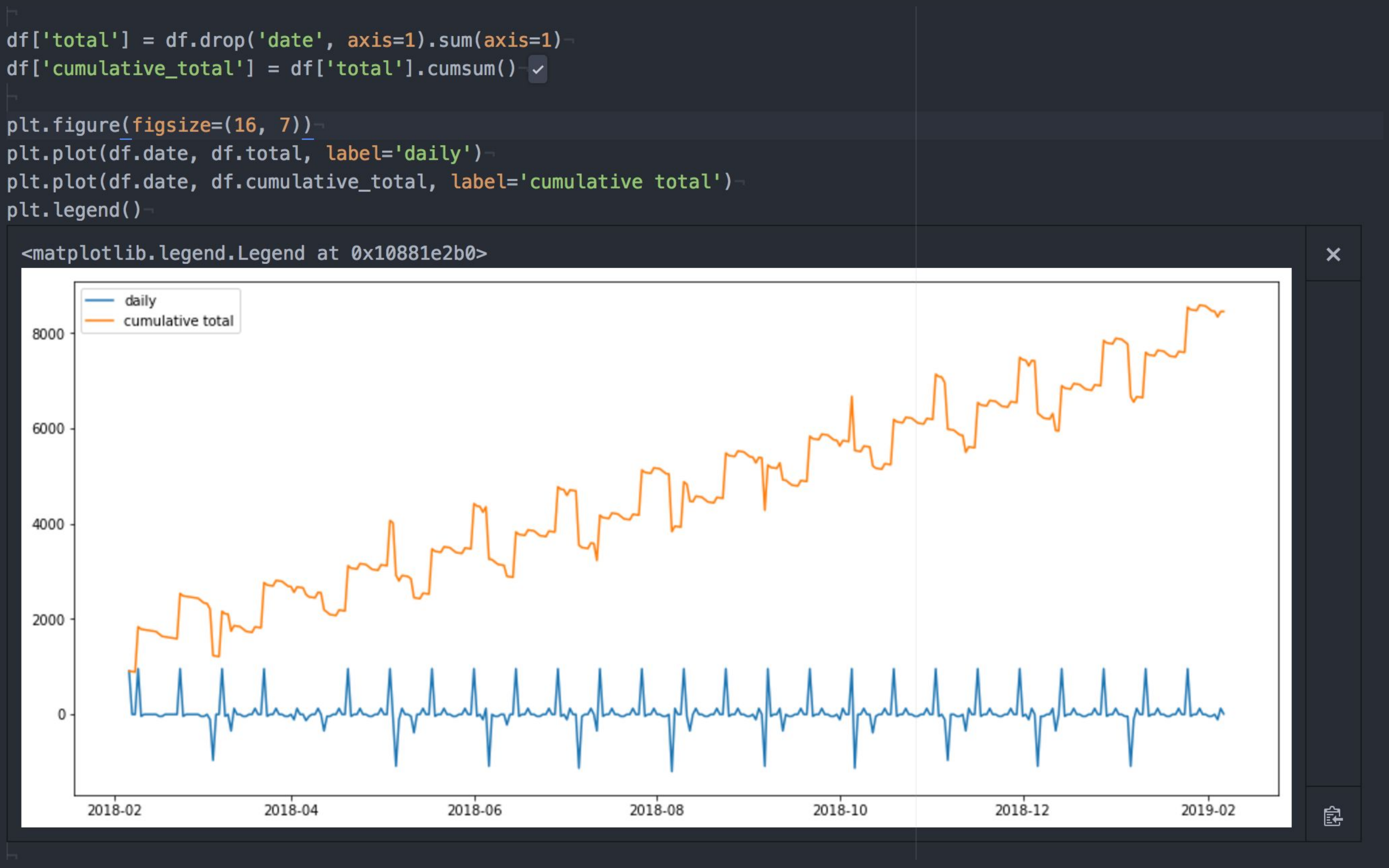
```
with open('data/inputs.yaml', 'r') as f:
    inputs = yaml.load(f)

for k, v in inputs.items():
    frequency = v.get('frequency')
    amount = v.get('amount')
    rr = get_rrule_or_datetime(frequency)
    if type(rr) is datetime.datetime:
        date = normalize_datetime(rr)
        dfi = pd.DataFrame({'date': [date], k: [amount]})
    else:
        dates = rr.between(TODAY, END)
        dates = [normalize_datetime(d) for d in dates]
        dfi = pd.DataFrame({'date': dates, k: amount})
    df = df.merge(dfi, how='left').fillna(0)
```



	date	mining_income	bank	salary	loan	rent	utilities	food	fun	vacation
0	2018-02-06	0.0	2000.2	0.0	0.0	-1090.0	0.0	-10.0	0.0	0.0
1	2018-02-07	0.0	0.0	0.0	0.0	0.0	0.0	-10.0	0.0	0.0
2	2018-02-08	0.0	0.0	0.0	0.0	0.0	0.0	-10.0	0.0	0.0
3	2018-02-09	0.0	0.0	1000.0	0.0	0.0	0.0	-10.0	-40.0	0.0
4	2018-02-10	0.0	0.0	0.0	0.0	0.0	0.0	-10.0	-40.0	0.0
5	2018-02-11	0.0	0.0	0.0	0.0	0.0	0.0	-10.0	0.0	0.0
6	2018-02-12	0.0	0.0	0.0	0.0	0.0	0.0	-10.0	0.0	0.0
7	2018-02-13	0.0	0.0	0.0	0.0	0.0	0.0	-10.0	0.0	0.0
8	2018-02-14	0.0	0.0	0.0	0.0	0.0	0.0	-10.0	0.0	0.0
9	2018-02-15	0.0	0.0	0.0	0.0	0.0	0.0	-10.0	0.0	0.0
10	2018-02-16	0.0	0.0	0.0	0.0	0.0	0.0	-10.0	-40.0	0.0



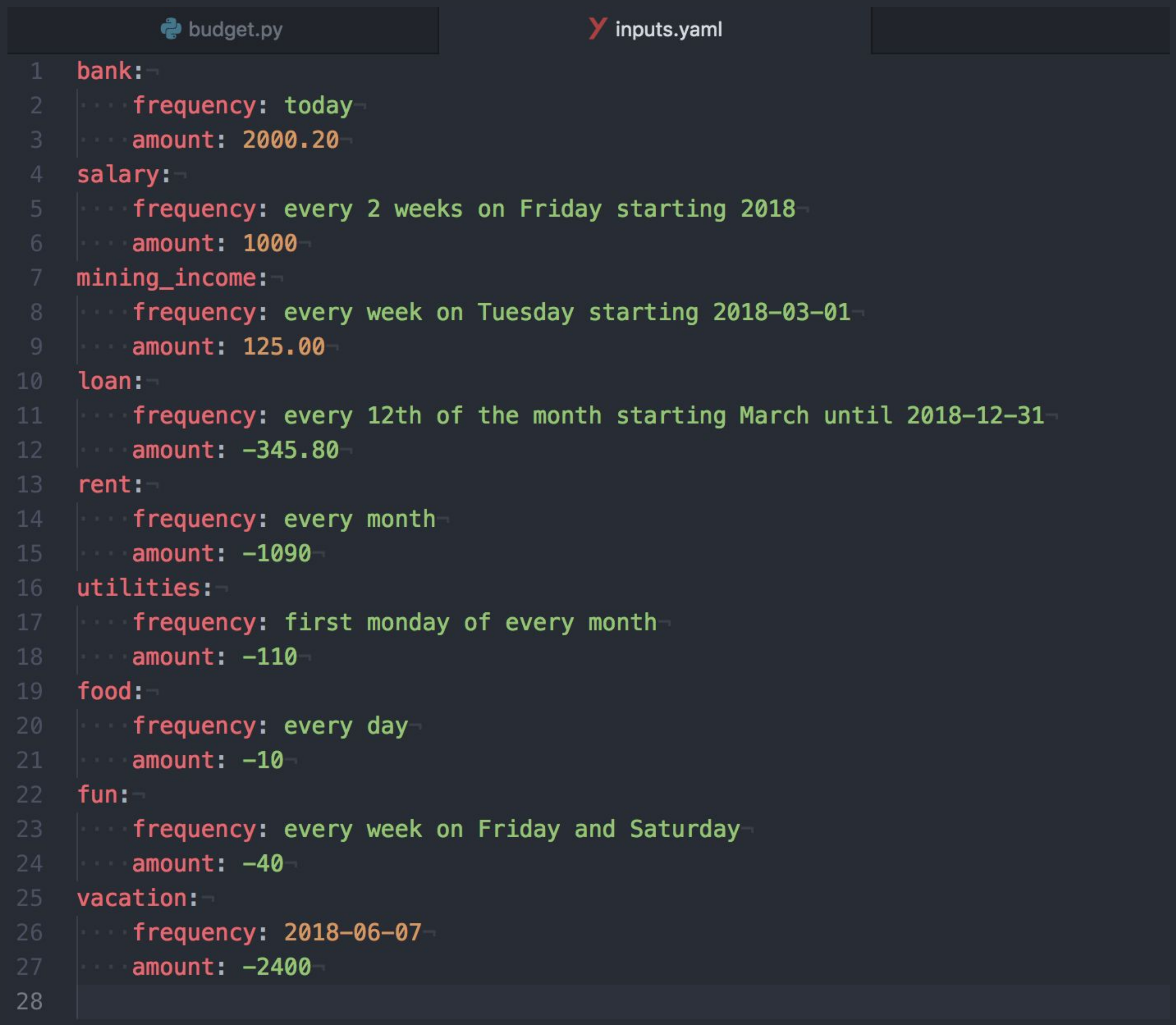
```
df['total'] = df.drop('date', axis=1).sum(axis=1)
df['cumulative_total'] = df['total'].cumsum()

plt.figure(figsize=(16, 7))
plt.plot(df.date, df.total, label='daily')
plt.plot(df.date, df.cumulative_total, label='cumulative total')
plt.legend()
```



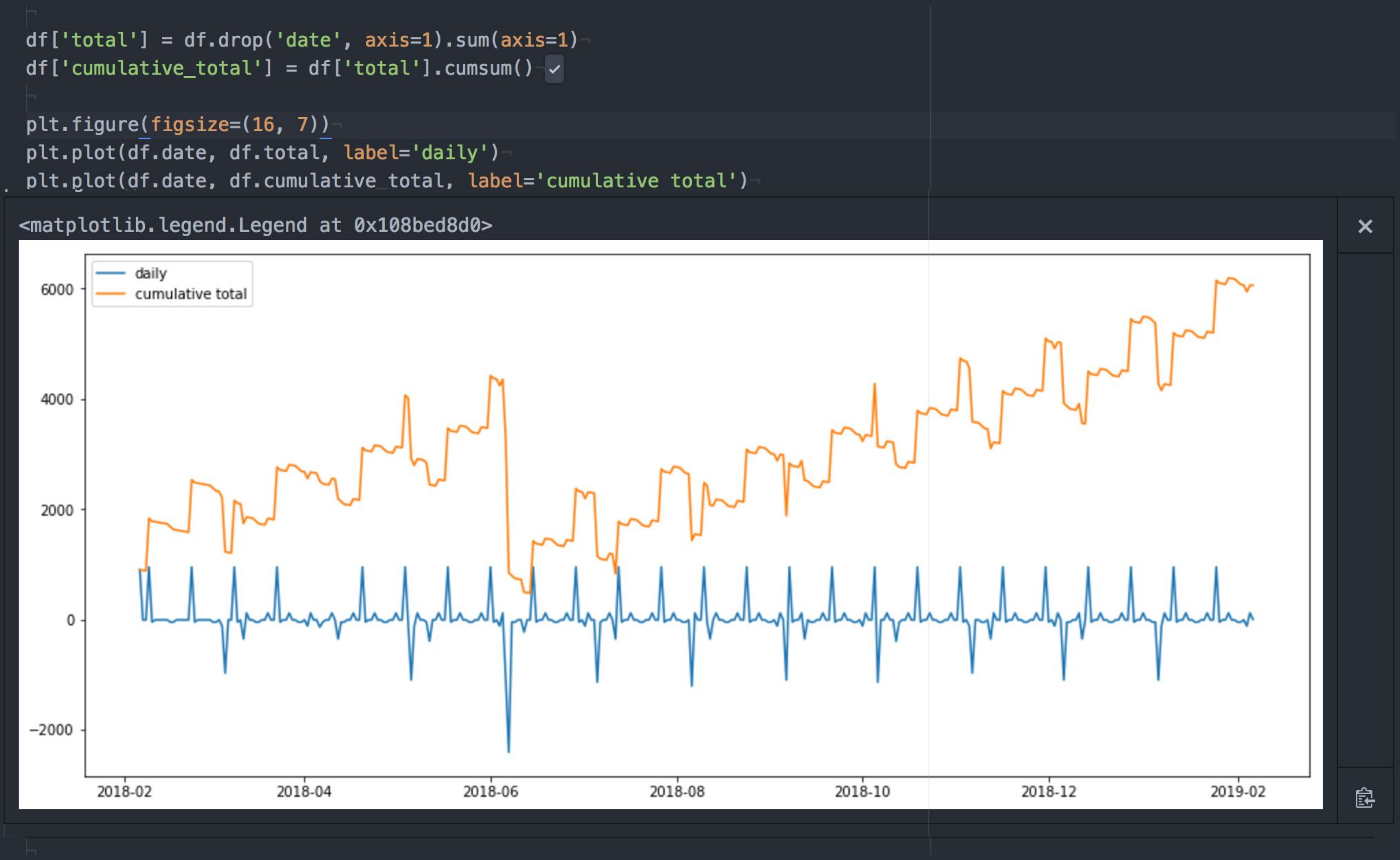
```
df['total'] = df.drop('date', axis=1).sum(axis=1)
df['cumulative_total'] = df['total'].cumsum()

plt.figure(figsize=(16, 7))
plt.plot(df.date, df.total, label='daily')
plt.plot(df.date, df.cumulative_total, label='cumulative total')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x10881e2b0>
```

I NEED A VACATION.

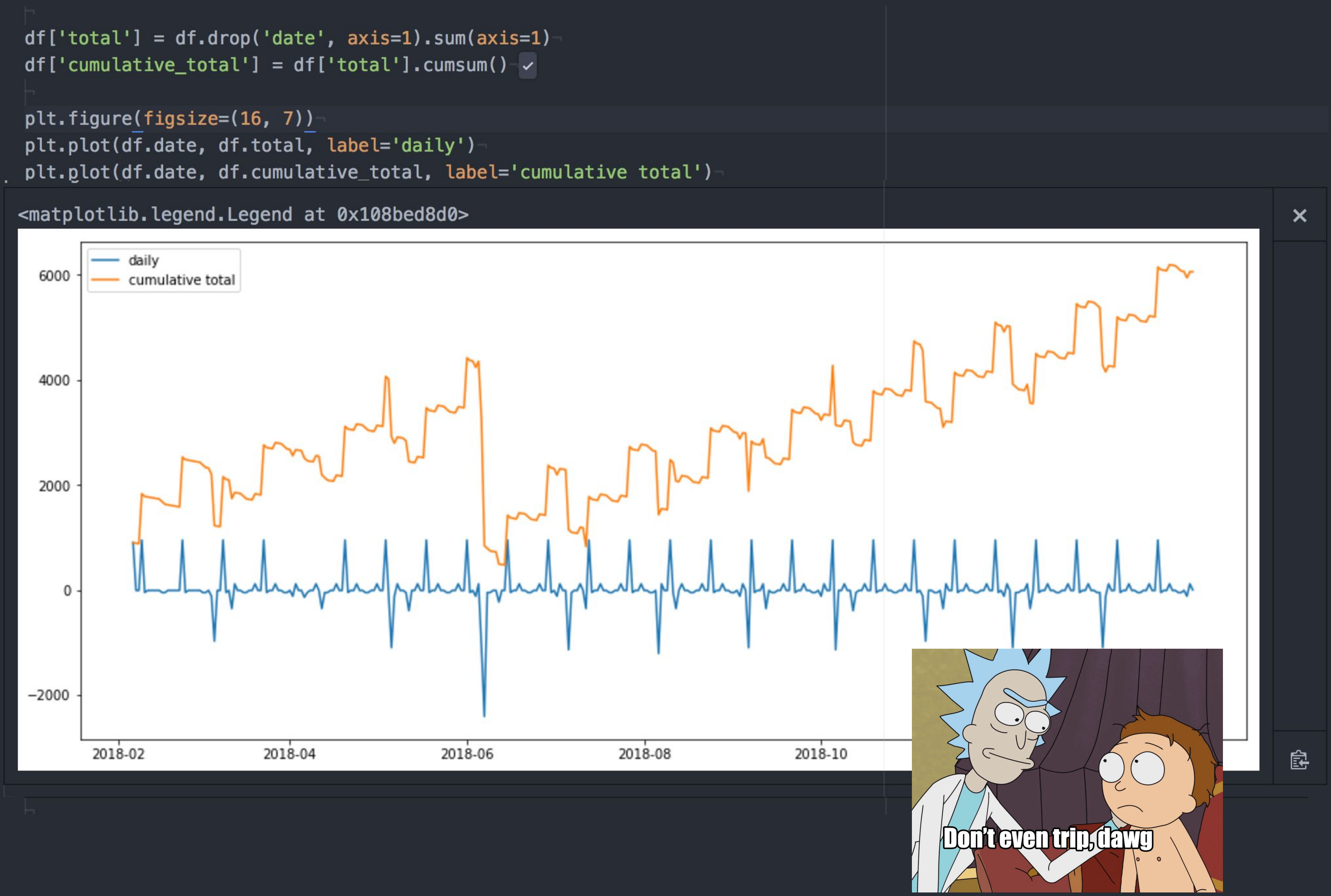


```yaml
bank:
    frequency: today
    amount: 2000.20
salary:
    frequency: every 2 weeks on Friday starting 2018
    amount: 1000
mining_income:
    frequency: every week on Tuesday starting 2018-03-01
    amount: 125.00
loan:
    frequency: every 12th of the month starting March until 2018-12-31
    amount: -345.80
rent:
    frequency: every month
    amount: -1090
utilities:
    frequency: first monday of every month
    amount: -110
food:
    frequency: every day
    amount: -10
fun:
    frequency: every week on Friday and Saturday
    amount: -40
vacation:
    frequency: 2018-06-07
    amount: -2400
```



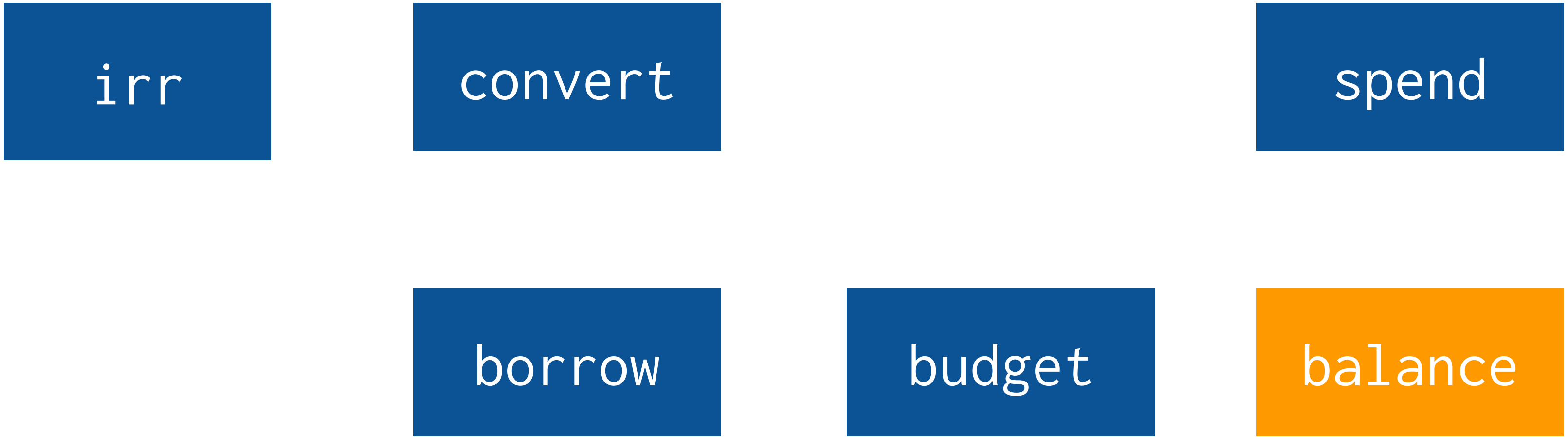
```
df['total'] = df.drop('date', axis=1).sum(axis=1)
df['cumulative_total'] = df['total'].cumsum()

plt.figure(figsize=(16, 7))
plt.plot(df.date, df.total, label='daily')
plt.plot(df.date, df.cumulative_total, label='cumulative total')
```

```
<matplotlib.legend.Legend at 0x108bed8d0>
```



```
df['total'] = df.drop('date', axis=1).sum(axis=1)
df['cumulative_total'] = df['total'].cumsum()

plt.figure(figsize=(16, 7))
plt.plot(df.date, df.total, label='daily')
plt.plot(df.date, df.cumulative_total, label='cumulative total')
```

```
<matplotlib.legend.Legend at 0x108bed8d0>
```



irr

convert

spend

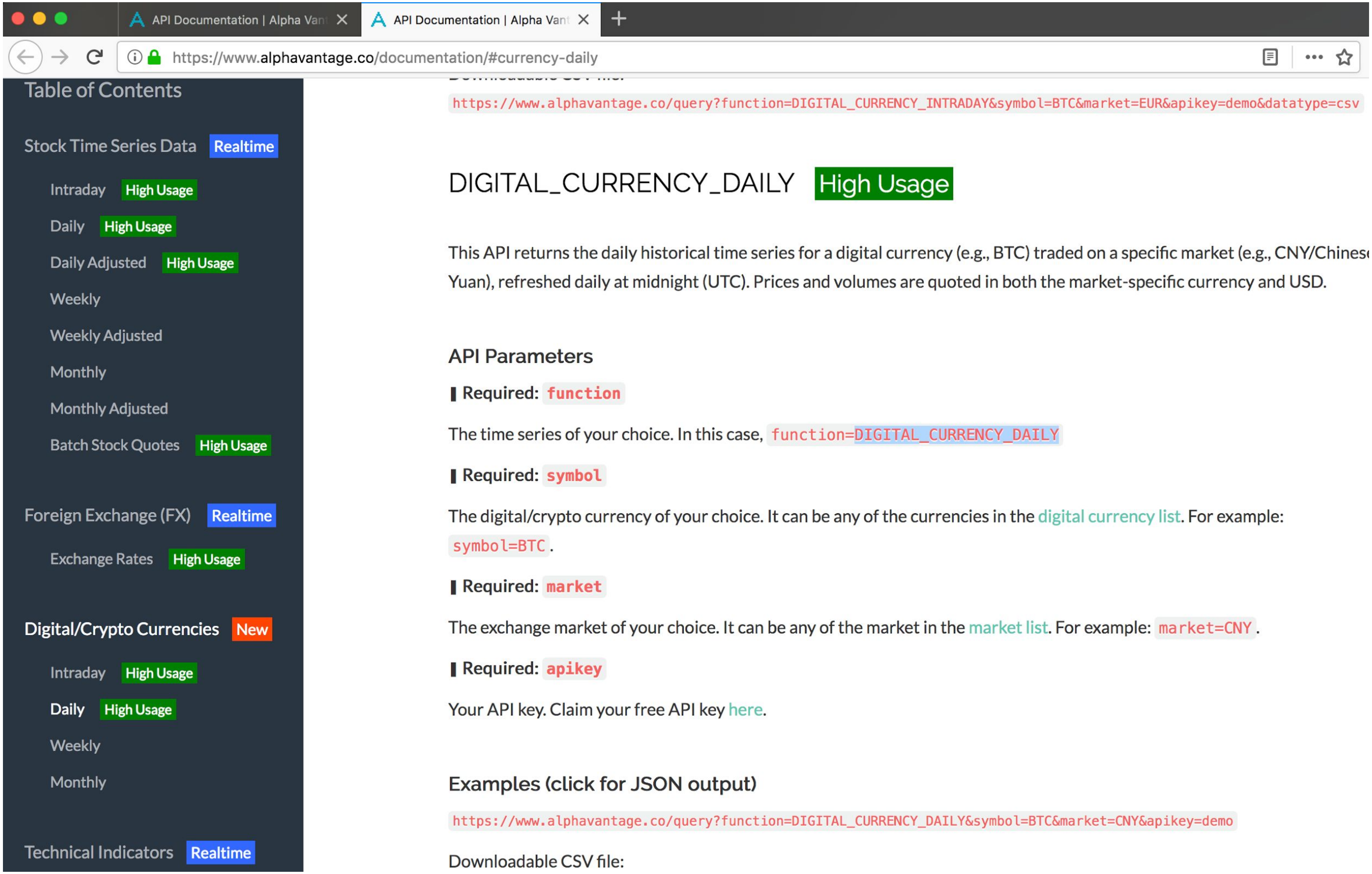
borrow

budget

balance

coins
shiba
You need to diversify your [redacted], [redacted]

ethereum


API Documentation | Alpha Van
API Documentation | Alpha Vant
https://www.alphavantage.co/documentation/#currency-daily

Downloadable CSV file:

`https://www.alphavantage.co/query?function=DIGITAL_CURRENCY_INTRADAY&symbol=BTC&market=EUR&apikey=demo&datatype=csv`

DIGITAL_CURRENCY_DAILY High Usage

This API returns the daily historical time series for a digital currency (e.g., BTC) traded on a specific market (e.g., CNY/Chinese Yuan), refreshed daily at midnight (UTC). Prices and volumes are quoted in both the market-specific currency and USD.

API Parameters

❚ **Required:** `function`

The time series of your choice. In this case, `function=DIGITAL_CURRENCY_DAILY`

❚ **Required:** `symbol`

The digital/crypto currency of your choice. It can be any of the currencies in the digital currency list. For example: `symbol=BTC`.

❚ **Required:** `market`

The exchange market of your choice. It can be any of the market in the market list. For example: `market=CNY`.

❚ **Required:** `apikey`

Your API key. Claim your free API key here.

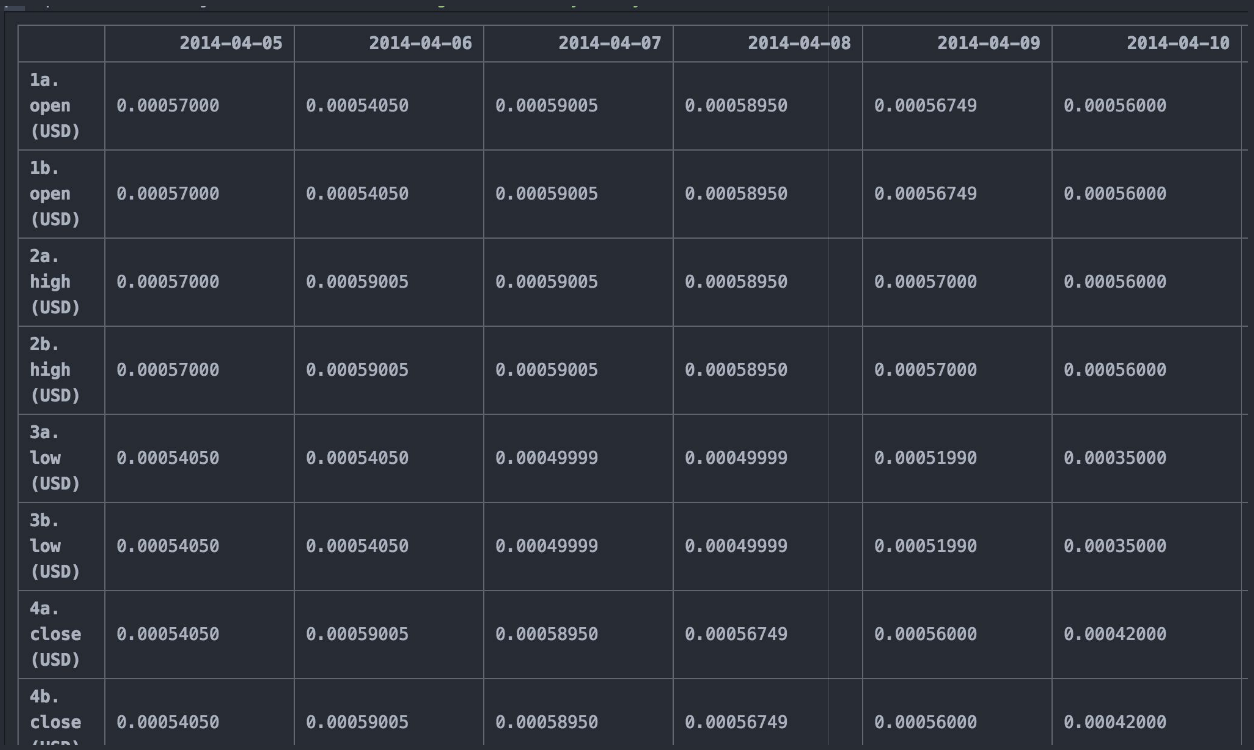
Examples (click for JSON output)

`https://www.alphavantage.co/query?function=DIGITAL_CURRENCY_DAILY&symbol=BTC&market=CNY&apikey=demo`

Downloadable CSV file:

```
URL = 'https://www.alphavantage.co/query?'
payload = {
    'function': 'DIGITAL_CURRENCY_DAILY',
    'symbol': ticker,
    'market': market,
    'apikey': API_KEY
}
r = requests.get(URL, params=payload)
```

```python
p = pd.DataFrame(r.json()['Time Series (Digital Currency Daily)'])
```

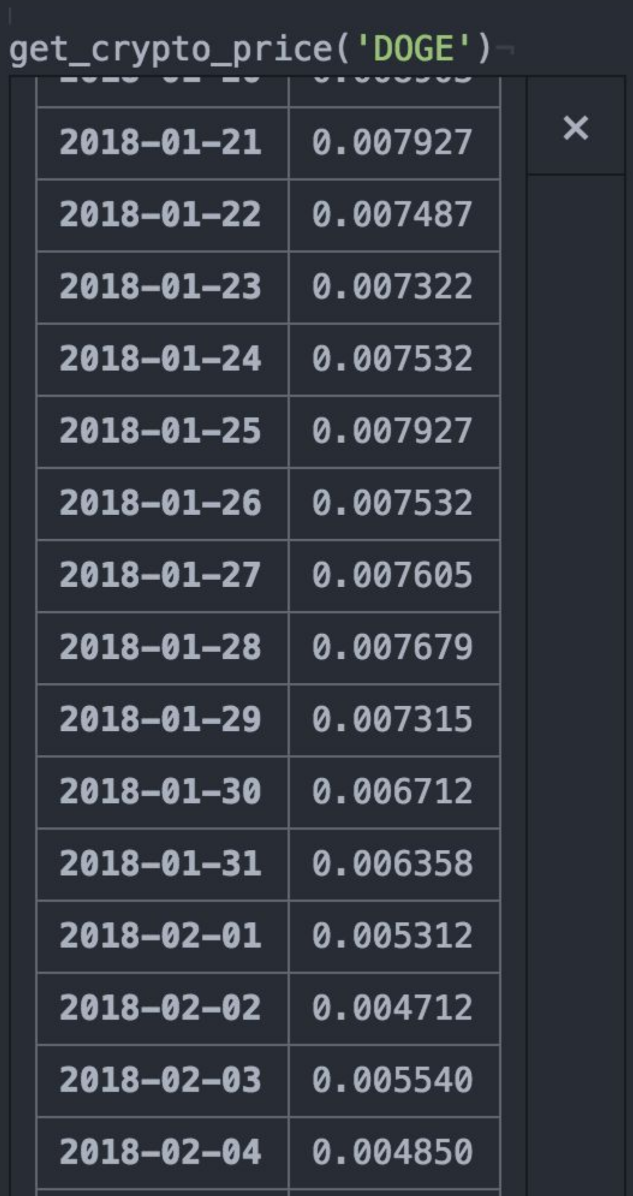


	2014-04-05	2014-04-06	2014-04-07	2014-04-08	2014-04-09	2014-04-10
1a. open (USD)	0.00057000	0.00054050	0.00059005	0.00058950	0.00056749	0.00056000
1b. open (USD)	0.00057000	0.00054050	0.00059005	0.00058950	0.00056749	0.00056000
2a. high (USD)	0.00057000	0.00059005	0.00059005	0.00058950	0.00057000	0.00056000
2b. high (USD)	0.00057000	0.00059005	0.00059005	0.00058950	0.00057000	0.00056000
3a. low (USD)	0.00054050	0.00054050	0.00049999	0.00049999	0.00051990	0.00035000
3b. low (USD)	0.00054050	0.00054050	0.00049999	0.00049999	0.00051990	0.00035000
4a. close (USD)	0.00054050	0.00059005	0.00058950	0.00056749	0.00056000	0.00042000
4b. close (USD)	0.00054050	0.00059005	0.00058950	0.00056749	0.00056000	0.00042000

```
p = p.T['4a. close (USD)']
```

```
2014-04-05    0.00054050
2014-04-06    0.00059005
2014-04-07    0.00058950
2014-04-08    0.00056749
2014-04-09    0.00056000
2014-04-10    0.00042000
2014-04-11    0.00050000
2014-04-12    0.00054700
2014-04-13    0.00045000
2014-04-14    0.00050000
2014-04-15    0.00056000
2014-04-16    0.00076001
2014-04-17    0.00070000
2014-04-18    0.00065000
2014-04-19    0.00070000
2014-04-20    0.00068000
2014-04-21    0.00068000
2014-04-22    0.00070000
2014-04-23    0.00062610
2014-04-24    0.00065994
```

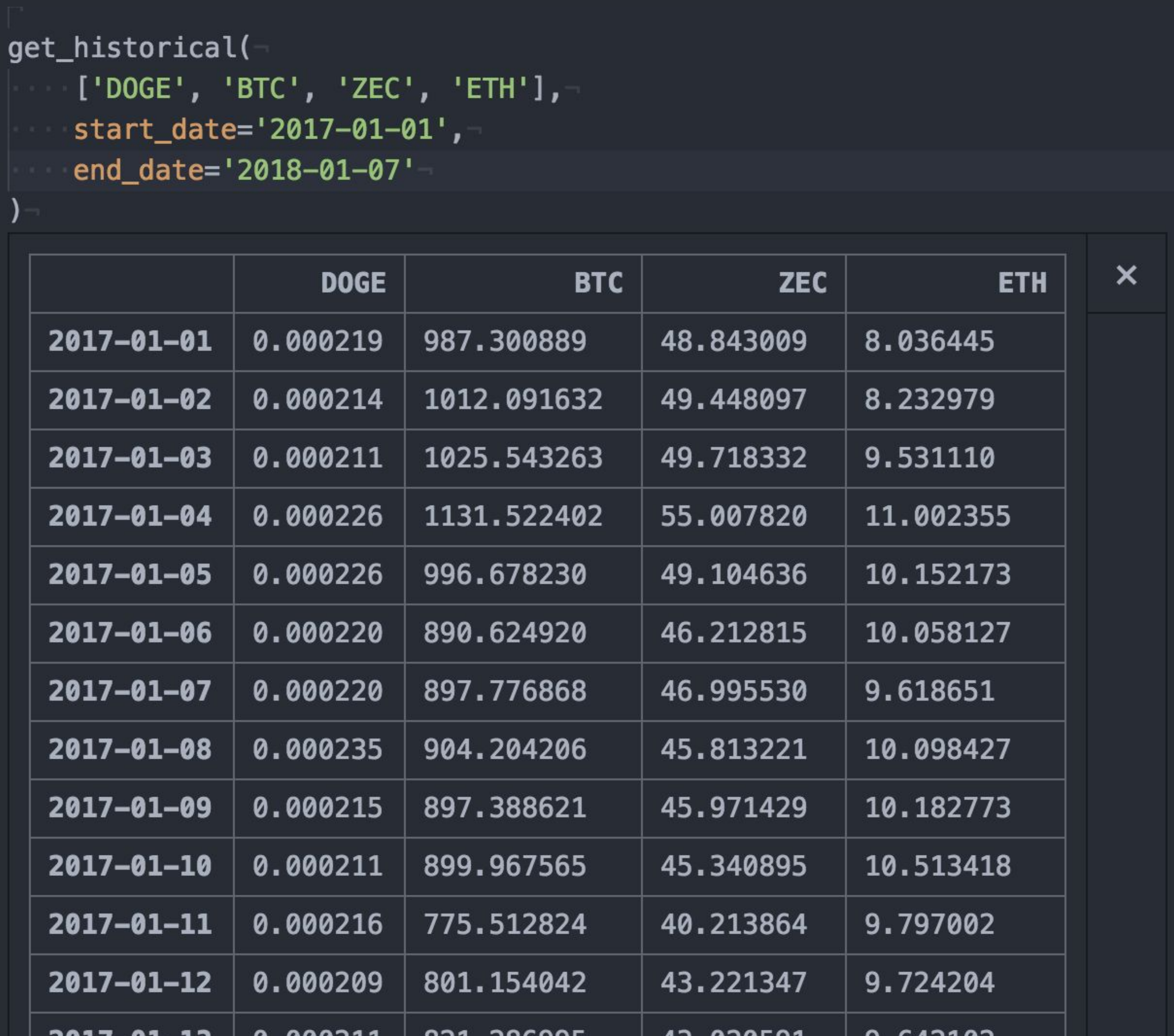
```
def get_crypto_price(ticker, market='USD', latest=False):
    URL = 'https://www.alphavantage.co/query?'
    payload = {
        'function': 'DIGITAL_CURRENCY_DAILY',
        'symbol': ticker,
        'market': market,
        'apikey': API_KEY
    }
    r = requests.get(URL, params=payload)
    p = pd.DataFrame(
              r.json()['Time Series (Digital Currency Daily)’])
             .T[‘4a. close (USD)']
    df = pd.DataFrame({ticker: p.apply(float)})
    df.index = pd.to_datetime(df.index)
    if latest:
        return df.tail(1)
    return df
```



```
get_crypto_price('DOGE')
```

```
2018-01-21   0.007927
2018-01-22   0.007487
2018-01-23   0.007322
2018-01-24   0.007532
2018-01-25   0.007927
2018-01-26   0.007532
2018-01-27   0.007605
2018-01-28   0.007679
2018-01-29   0.007315
2018-01-30   0.006712
2018-01-31   0.006358
2018-02-01   0.005312
2018-02-02   0.004712
2018-02-03   0.005540
2018-02-04   0.004850
```

```python
def get_historical(tickers, start_date, end_date):
    df = pd.DataFrame(
          index=pd.date_range(start_date, end_date, freq='D'))
    for t in tickers:
        df = pd.concat([
            df,
            get_crypto_price(t)],
            axis=1,
            join_axes=[df.index]
        )
    df = df.fillna(method='ffill').dropna()
    return df
```

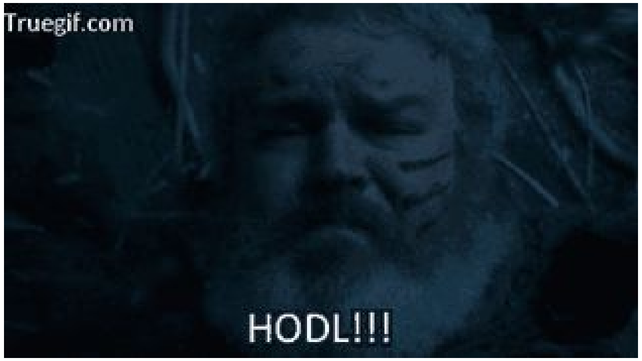


```
get_historical(
    ['DOGE', 'BTC', 'ZEC', 'ETH'],
    start_date='2017-01-01',
    end_date='2018-01-07'
)
```

	DOGE	BTC	ZEC	ETH
2017-01-01	0.000219	987.300889	48.843009	8.036445
2017-01-02	0.000214	1012.091632	49.448097	8.232979
2017-01-03	0.000211	1025.543263	49.718332	9.531110
2017-01-04	0.000226	1131.522402	55.007820	11.002355
2017-01-05	0.000226	996.678230	49.104636	10.152173
2017-01-06	0.000220	890.624920	46.212815	10.058127
2017-01-07	0.000220	897.776868	46.995530	9.618651
2017-01-08	0.000235	904.204206	45.813221	10.098427
2017-01-09	0.000215	897.388621	45.971429	10.182773
2017-01-10	0.000211	899.967565	45.340895	10.513418
2017-01-11	0.000216	775.512824	40.213864	9.797002
2017-01-12	0.000209	801.154042	43.221347	9.724204
2017-01-13	0.000211	821.386005	43.030591	9.643103

```python
class Rebalance:

    def __init__(self, targets, deposit):

    def _instantiate_portfolio(self):

    def update_prices(self, prices):

    def get_order(self):

    def process_order(self):

    def deposit(self, amount):

    def withdraw(self, amount):
```

```
class Rebalance:

    def __init__(self, targets, deposit):

    def _instantiate_portfolio(self):

    def update_prices(self, prices):

    def get_order(self):

    def process_order(self):

    def deposit(self, amount):
```

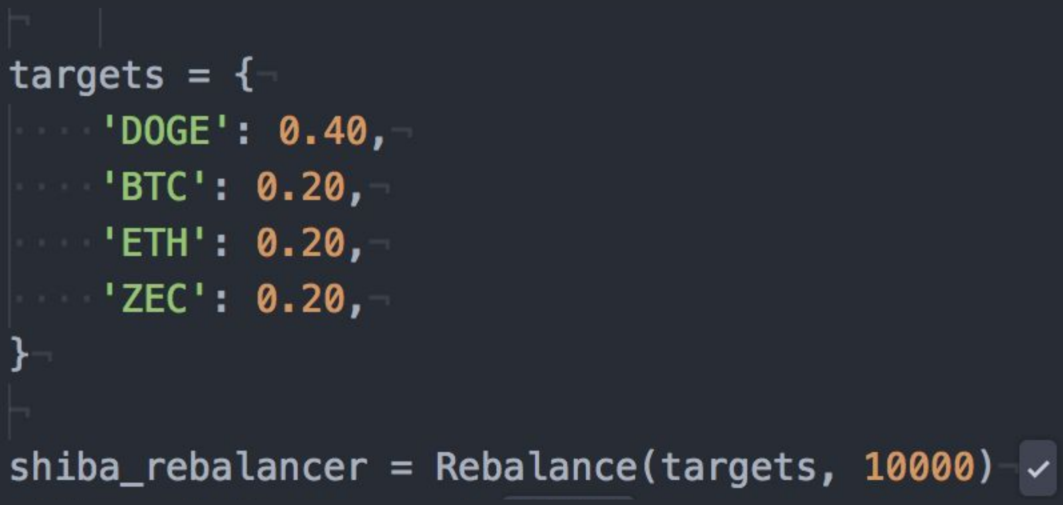


```
class Rebalance:

    def __init__(self, targets, deposit):

    def _instantiate_portfolio(self):

    def update_prices(self, prices):

    def get_order(self):

    def process_order(self):

    def deposit(self, amount):

    def withdraw(self, amount):
```

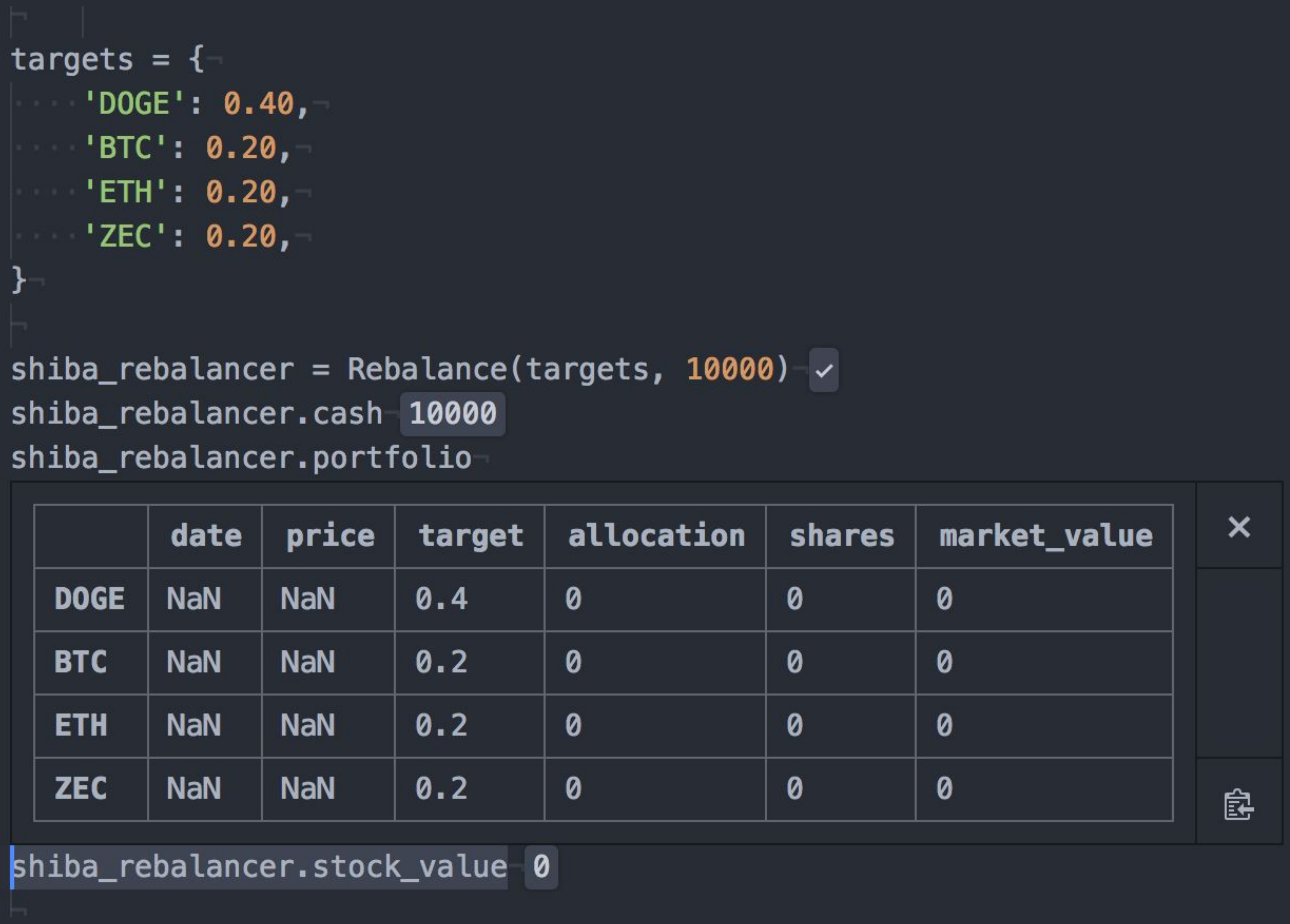
```python
def __init__(self, targets, deposit):
     self.targets = targets
     self.tickers = list(targets.keys())
     self.cash = deposit
     self.stock_value = 0
     self.total_value = self.cash + self.stock_value
     self.portfolio = self._instantiate_portfolio()
```

```python
class Rebalance:

    def __init__(self, targets, deposit):

    def _instantiate_portfolio(self):

    def update_prices(self, prices):

    def get_order(self):

    def process_order(self):

    def deposit(self, amount):

    def withdraw(self, amount):
```

```python
def _instantiate_portfolio(self):
     df = pd.DataFrame(
         index=self.tickers,
         columns=['date', 'price', 'target',
         ‘allocation', 'shares', 'market_value']
     )
     df.shares = 0
     df.market_value = 0
     df.allocation = 0
     df.update(
         pd.DataFrame
             .from_dict(self.targets, orient='index')
             .rename(columns={0:'target'})
     )
     return df
```



```python
targets = {
    'DOGE': 0.40,
    'BTC': 0.20,
    'ETH': 0.20,
    'ZEC': 0.20,
}

shiba_rebalancer = Rebalance(targets, 10000)
```



```
targets = {
    'DOGE': 0.40,
    'BTC': 0.20,
    'ETH': 0.20,
    'ZEC': 0.20,
}

shiba_rebalancer = Rebalance(targets, 10000) ✓
shiba_rebalancer.cash 10000
shiba_rebalancer.portfolio
```

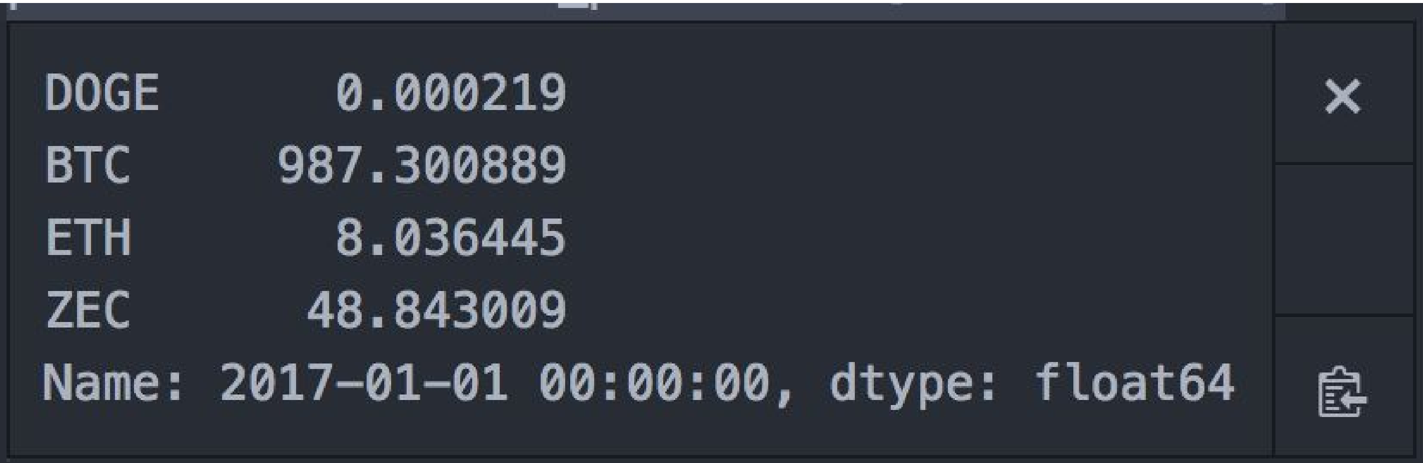
	date	price	target	allocation	shares	market_value
DOGE	NaN	NaN	0.4	0	0	0
BTC	NaN	NaN	0.2	0	0	0
ETH	NaN	NaN	0.2	0	0	0
ZEC	NaN	NaN	0.2	0	0	0

```
shiba_rebalancer.stock_value 0
```

```
class Rebalance:

    def __init__(self, targets, deposit):

    def _instantiate_portfolio(self):

    def update_prices(self, prices):

    def get_order(self):

    def process_order(self):

    def deposit(self, amount):

    def withdraw(self, amount):
```

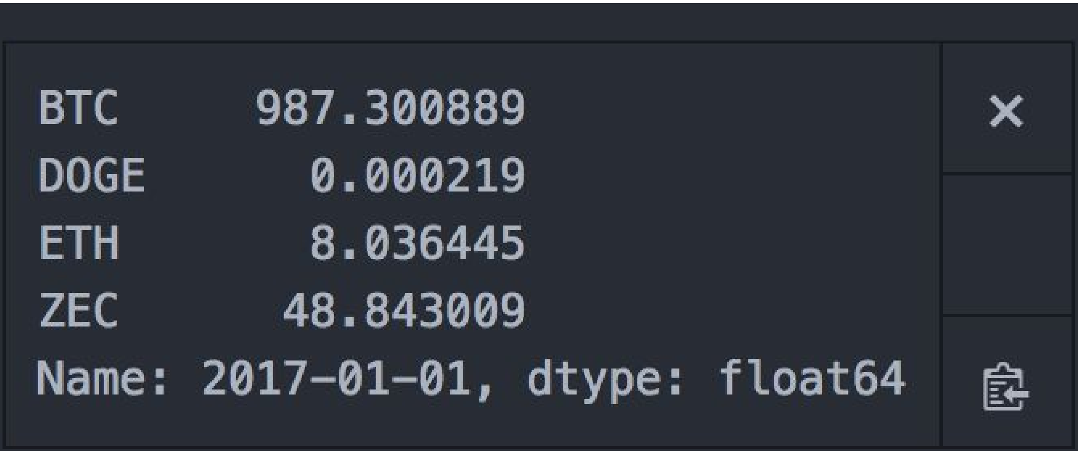
```
def update_prices(self, prices):
    self.portfolio.update(
        pd.DataFrame({
            'price': prices}
        )
    )
    self.portfolio.date = prices.name
    self.portfolio.market_value = (
        self.portfolio.shares * self.portfolio.price)
    self.stock_value = self.portfolio.market_value.sum()
    self.total_value = self.stock_value + self.cash
```

```python
tickers = list(targets.keys())
historical_prices = get_historical(
    tickers, '2017-01-01', '2018-01-07')
prices = historical_prices.loc['2017-01-01']
```



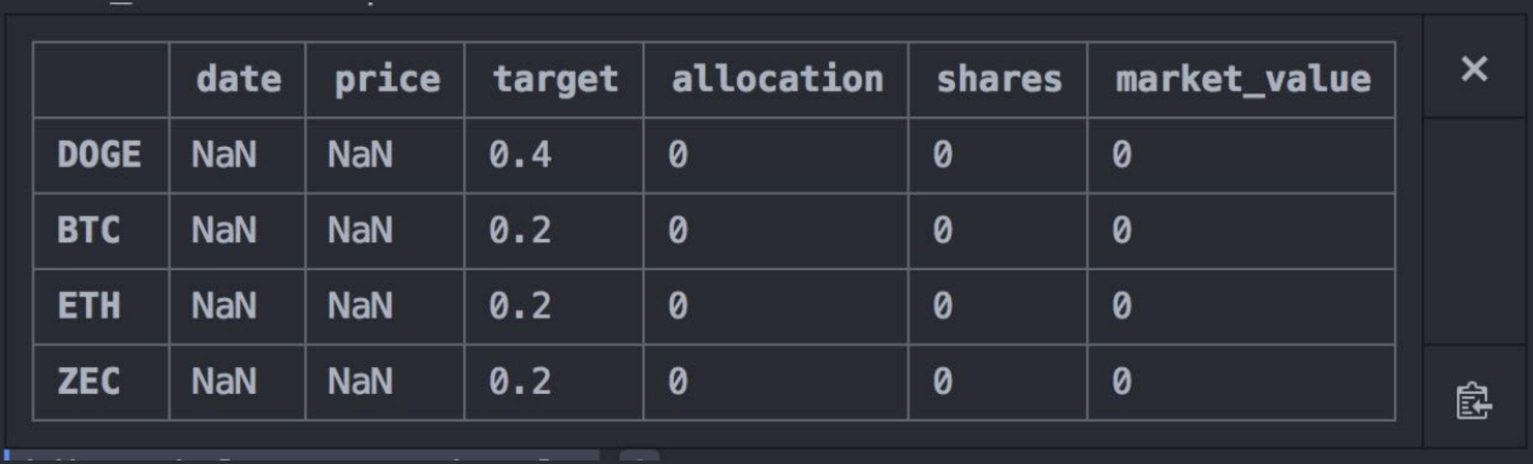
```
DOGE        0.000219
BTC       987.300889
ETH         8.036445
ZEC        48.843009
Name: 2017-01-01 00:00:00, dtype: float64
```

```python
prices = pd.Series({
    'DOGE': 0.000219,
    'BTC': 987.300889,
    'ETH': 8.036445,
    'ZEC': 48.843009
})
prices.name = '2017-01-01'
```



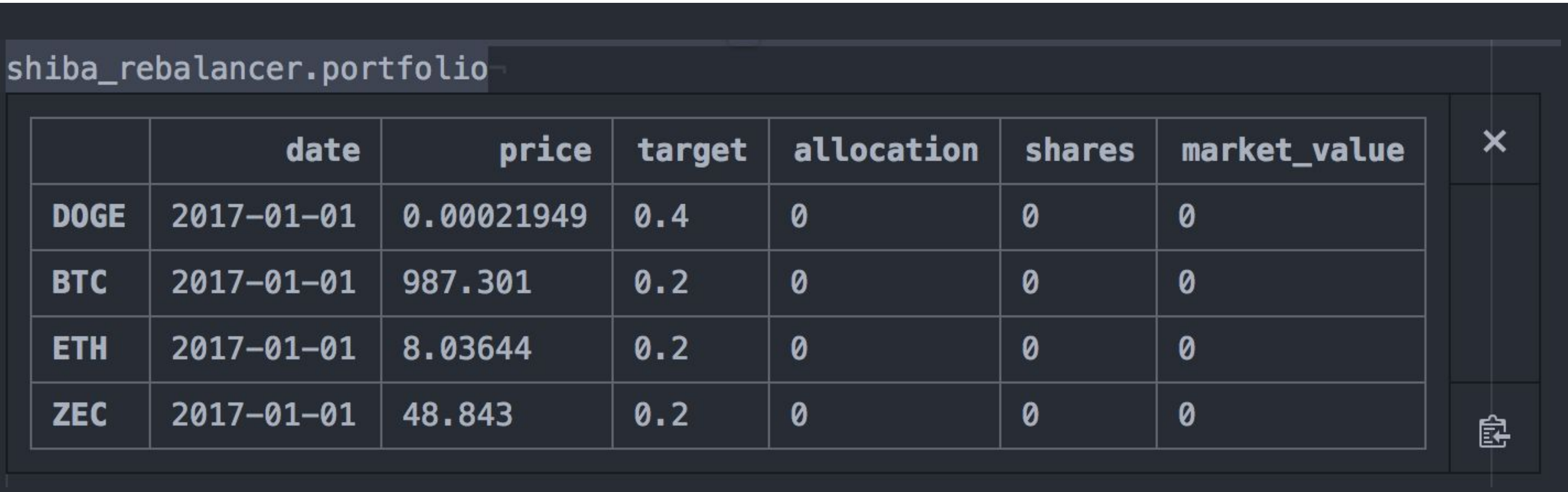
```
BTC      987.300889
DOGE       0.000219
ETH        8.036445
ZEC       48.843009
Name: 2017-01-01, dtype: float64
```

```
shiba_rebalancer = Rebalance(targets, 10000)
prices = historical_prices.loc['2017-01-01']
shiba_rebalancer.update_prices(prices)
```



	date	price	target	allocation	shares	market_value
DOGE	NaN	NaN	0.4	0	0	0
BTC	NaN	NaN	0.2	0	0	0
ETH	NaN	NaN	0.2	0	0	0
ZEC	NaN	NaN	0.2	0	0	0

```
shiba_rebalancer = Rebalance(targets, 10000)
prices = historical_prices.loc['2017-01-01']
shiba_rebalancer.update_prices(prices)
```

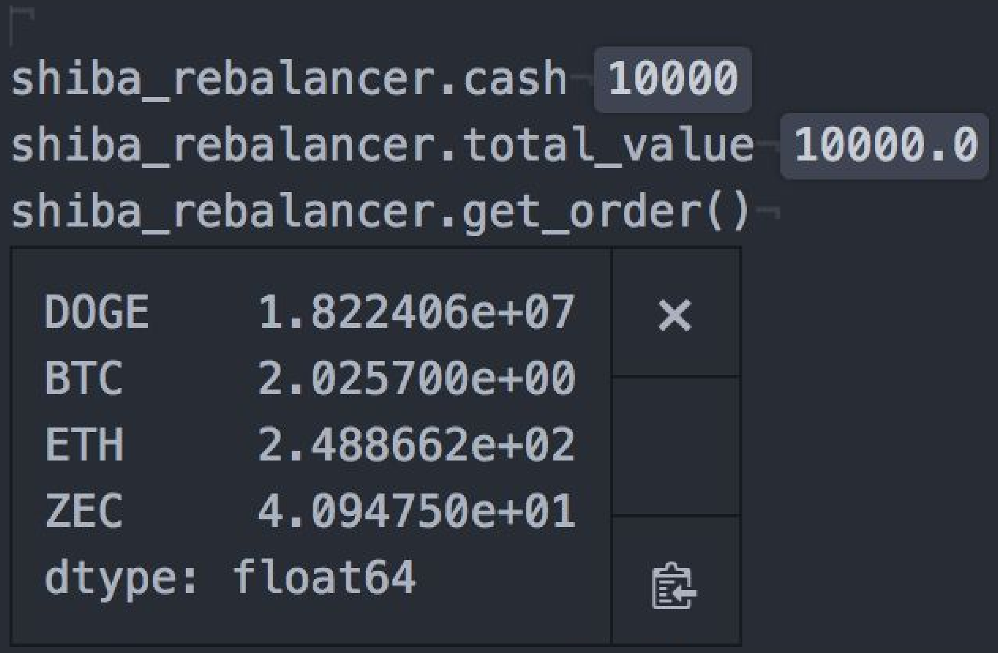


```
shiba_rebalancer.portfolio
```

	date	price	target	allocation	shares	market_value
DOGE	2017-01-01	0.00021949	0.4	0	0	0
BTC	2017-01-01	987.301	0.2	0	0	0
ETH	2017-01-01	8.03644	0.2	0	0	0
ZEC	2017-01-01	48.843	0.2	0	0	0

```python
class Rebalance:

    def __init__(self, targets, deposit):

    def _instantiate_portfolio(self):

    def update_prices(self, prices):

    def get_order(self):

    def process_order(self):

    def deposit(self, amount):

    def withdraw(self, amount):
```

```python
def get_order(self):
     self.order = (
         (self.total_value * self.portfolio.target
         / self.portfolio.price)
         - self.portfolio.shares
     ).apply(lambda x: safe_round_down(x, 4))
     print(self.order)
```

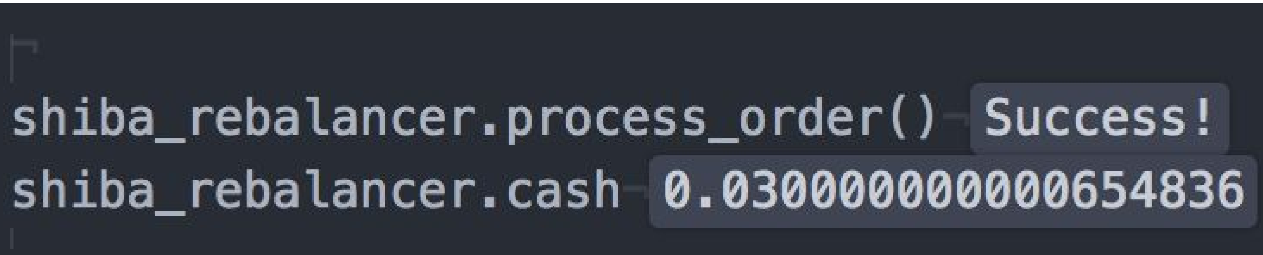


```
shiba_rebalancer.cash 10000
shiba_rebalancer.total_value 10000.0
shiba_rebalancer.get_order()

DOGE    1.822406e+07
BTC     2.025700e+00
ETH     2.488662e+02
ZEC     4.094750e+01
dtype: float64
```

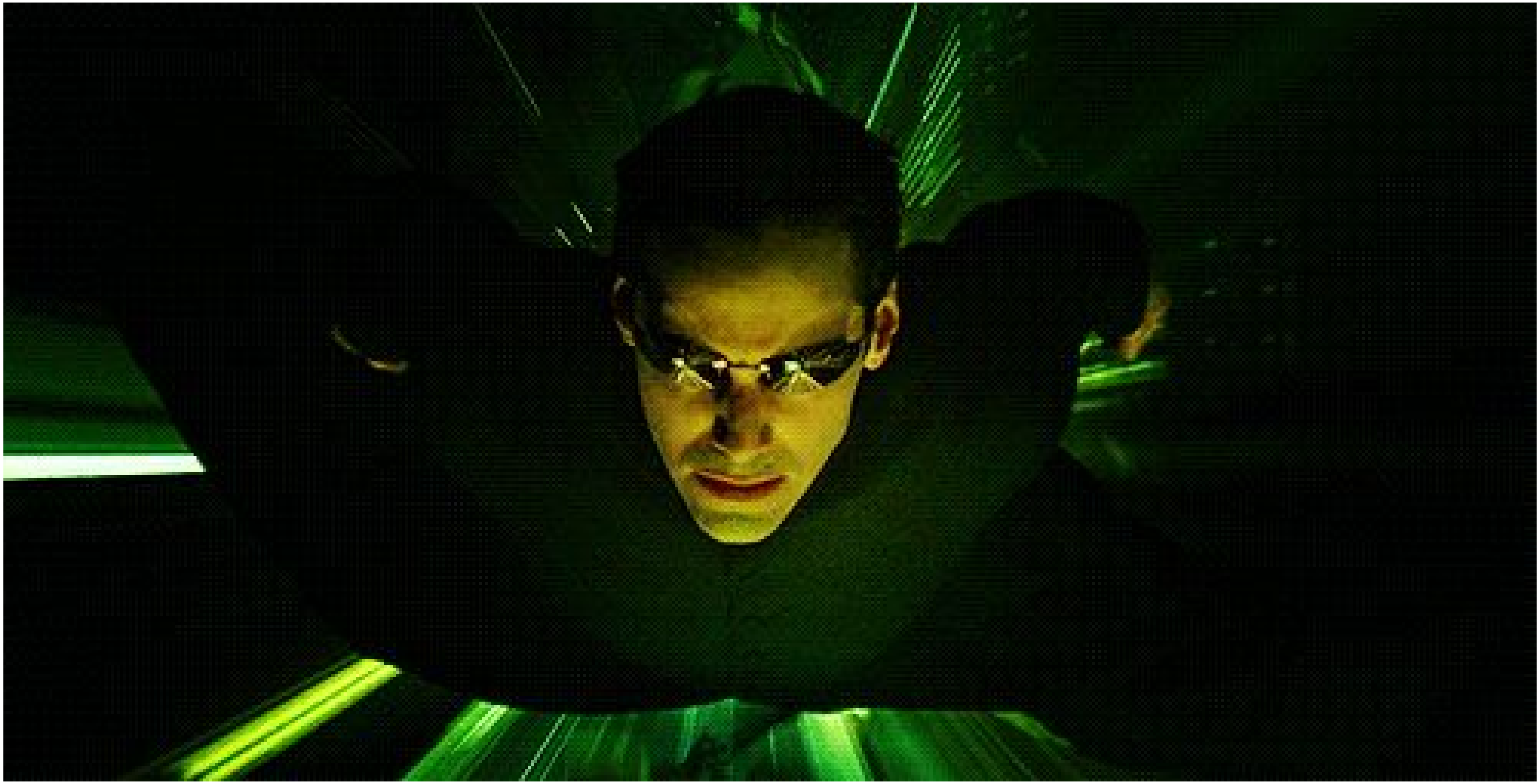
```
class Rebalance:

    def __init__(self, targets, deposit):

    def _instantiate_portfolio(self):

    def update_prices(self, prices):

    def get_order(self):

    def process_order(self):

    def deposit(self, amount):

    def withdraw(self, amount):
```

```python
def process_order(self):
    self.cash -= np.round(np.sum(self.order * self.portfolio.price), 2)
    self.portfolio.shares += self.order
    self.portfolio.market_value = self.portfolio.shares *
        self.portfolio.price
    self.portfolio.allocation = self.portfolio.market_value /
        self.total_value
    self.stock_value = self.portfolio.market_value.sum()
    self.total_value = self.cash + self.stock_value
    print('Success!')
```

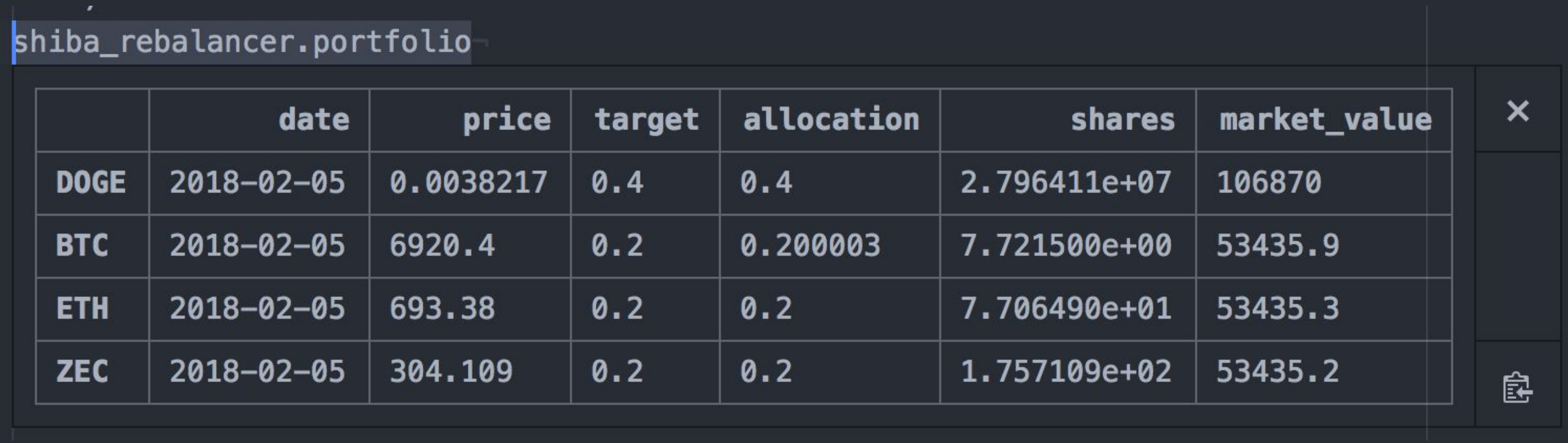
```python
def process_order(self):
     self.cash -= np.round(np.sum(self.order * self.portfolio.price), 2)
     self.portfolio.shares += self.order
     self.portfolio.market_value = self.portfolio.shares *
          self.portfolio.price
     self.portfolio.allocation = self.portfolio.market_value /
          self.total_value
     self.stock_value = self.portfolio.market_value.sum()
     self.total_value = self.cash + self.stock_value
     print('Success!')
```



```
shiba_rebalancer.process_order() Success!
shiba_rebalancer.cash 0.030000000000654836
```

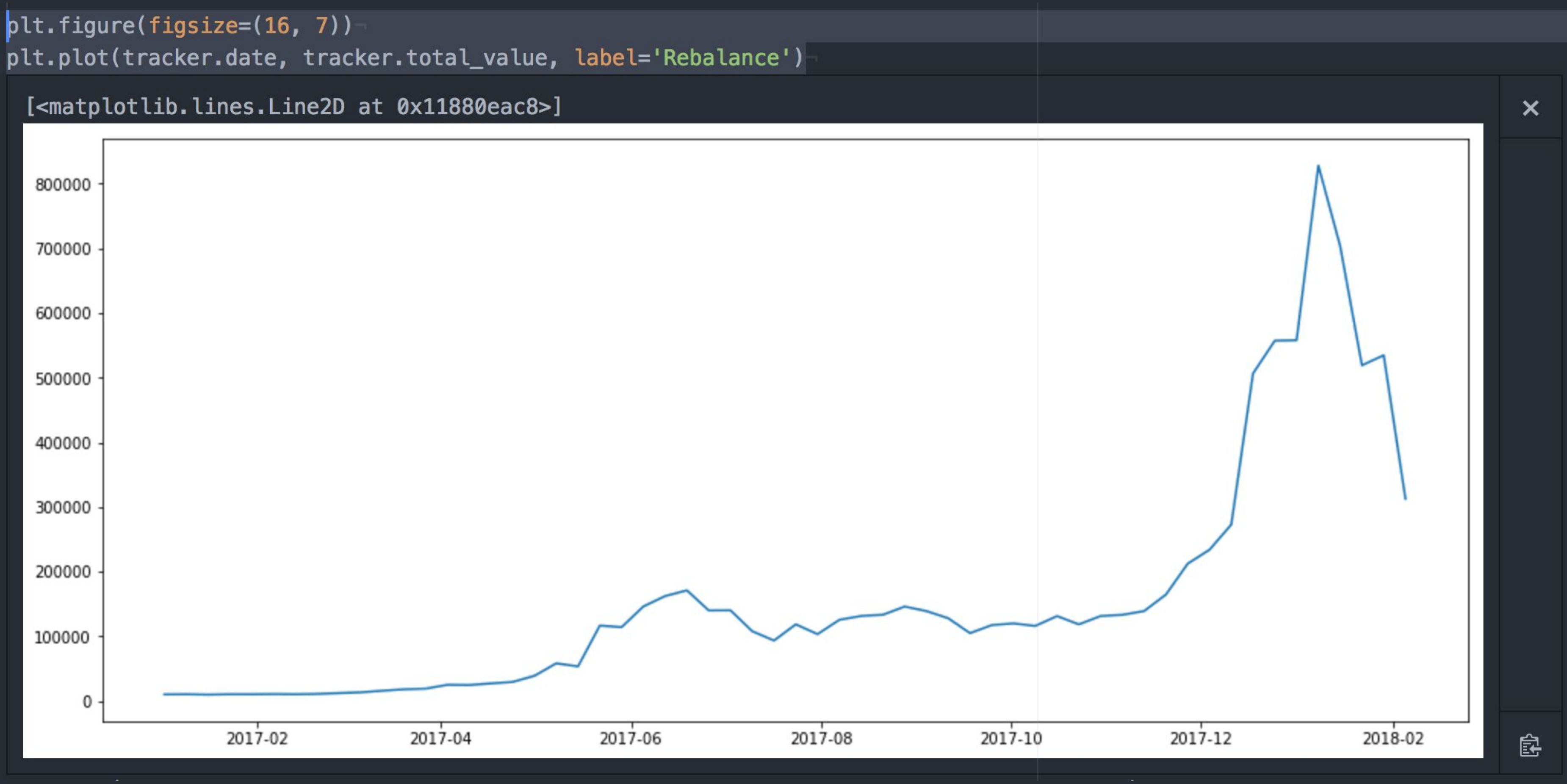


```
shiba_rebalancer = Rebalance(targets, 10000)
dates = pd.date_range(
   ‘2017-01-01', '2018-02-06', freq='W-MON').tolist()
tracker = pd.DataFrame()
for d in dates:
    prices = historical_prices.loc[d]
    shiba_rebalancer.update_prices(prices)
    shiba_rebalancer.get_order()
    shiba_rebalancer.process_order()
    tracker = tracker.append(
        pd.DataFrame({
            'date': [d],
            'total_value': [shiba_rebalancer.total_value]
        })
    )
```



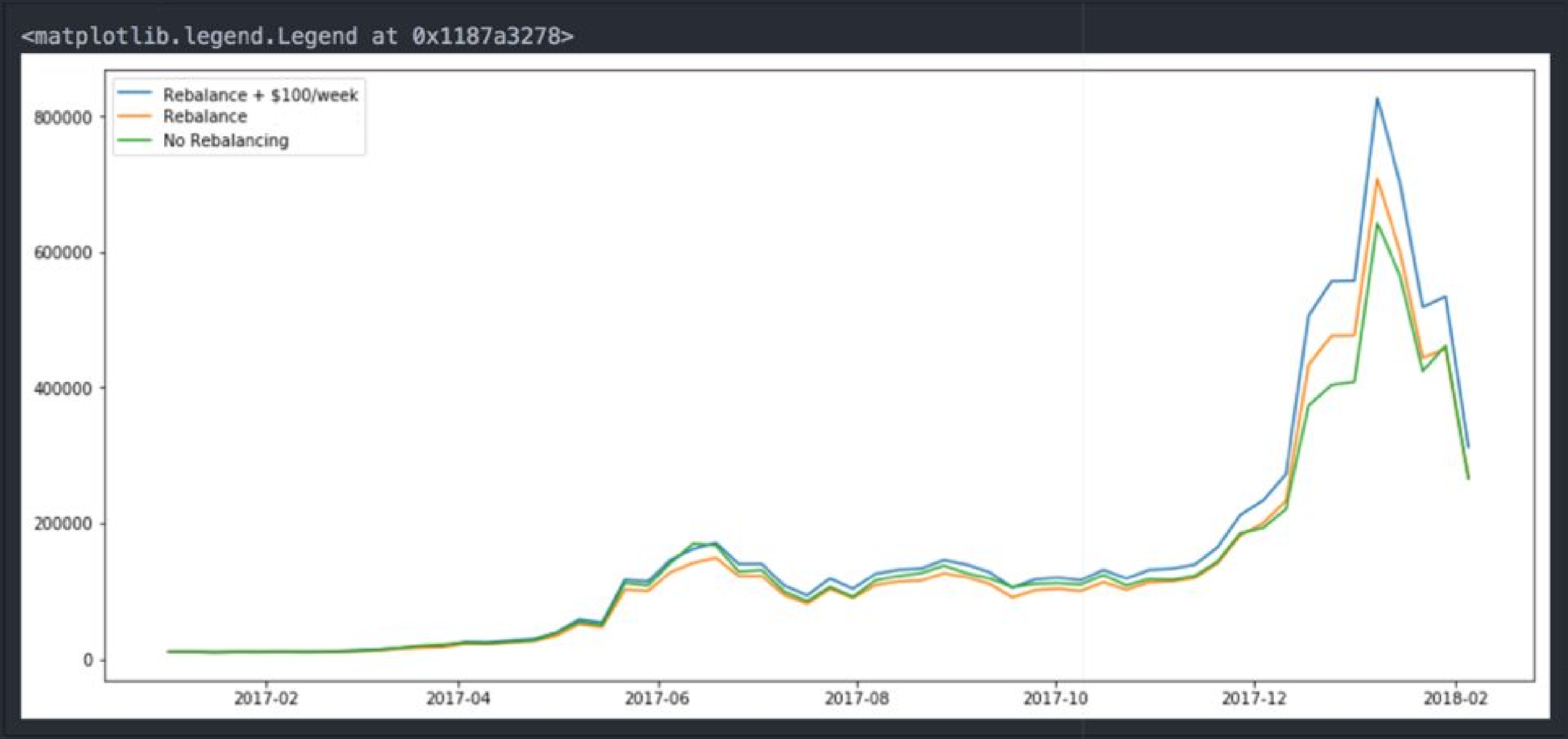
```
shiba_rebalancer.portfolio
```

	date	price	target	allocation	shares	market_value
DOGE	2018-02-05	0.0038217	0.4	0.4	2.796411e+07	106870
BTC	2018-02-05	6920.4	0.2	0.200003	7.721500e+00	53435.9
ETH	2018-02-05	693.38	0.2	0.2	7.706490e+01	53435.3
ZEC	2018-02-05	304.109	0.2	0.2	1.757109e+02	53435.2



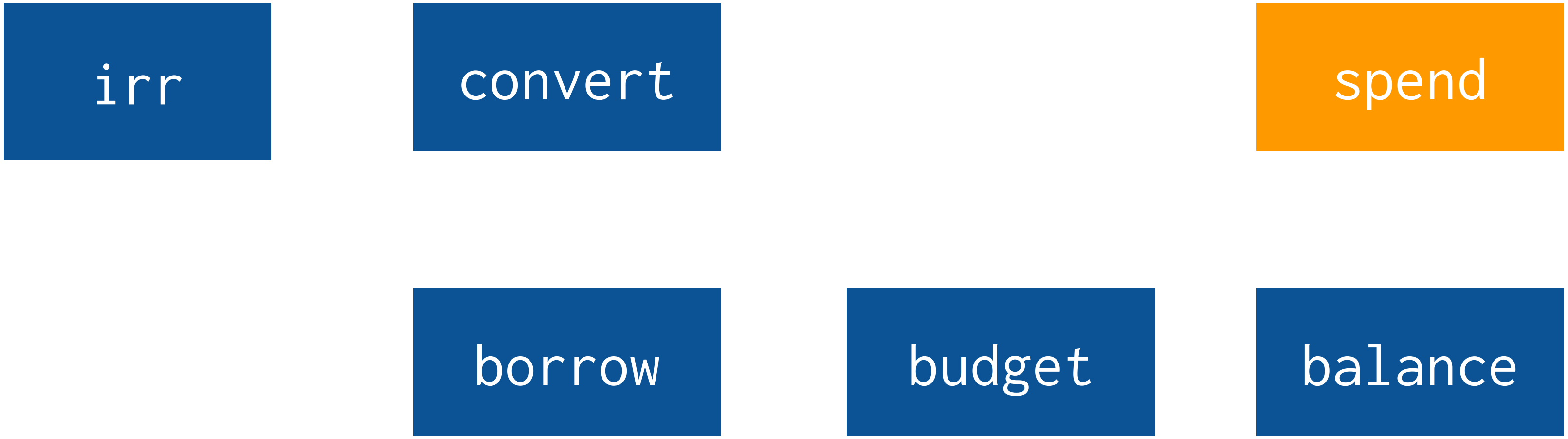
```
plt.figure(figsize=(16, 7))
plt.plot(tracker.date, tracker.total_value, label='Rebalance')
```

```
[<matplotlib.lines.Line2D at 0x11880eac8>]
```



```
<matplotlib.legend.Legend at 0x1187a3278>
```

Rebalance + $100/week
Rebalance
No Rebalancing
800000
600000
400000
200000
0
2017-02
2017-04
2017-06
2017-08
2017-10
2017-12
2018-02



irr

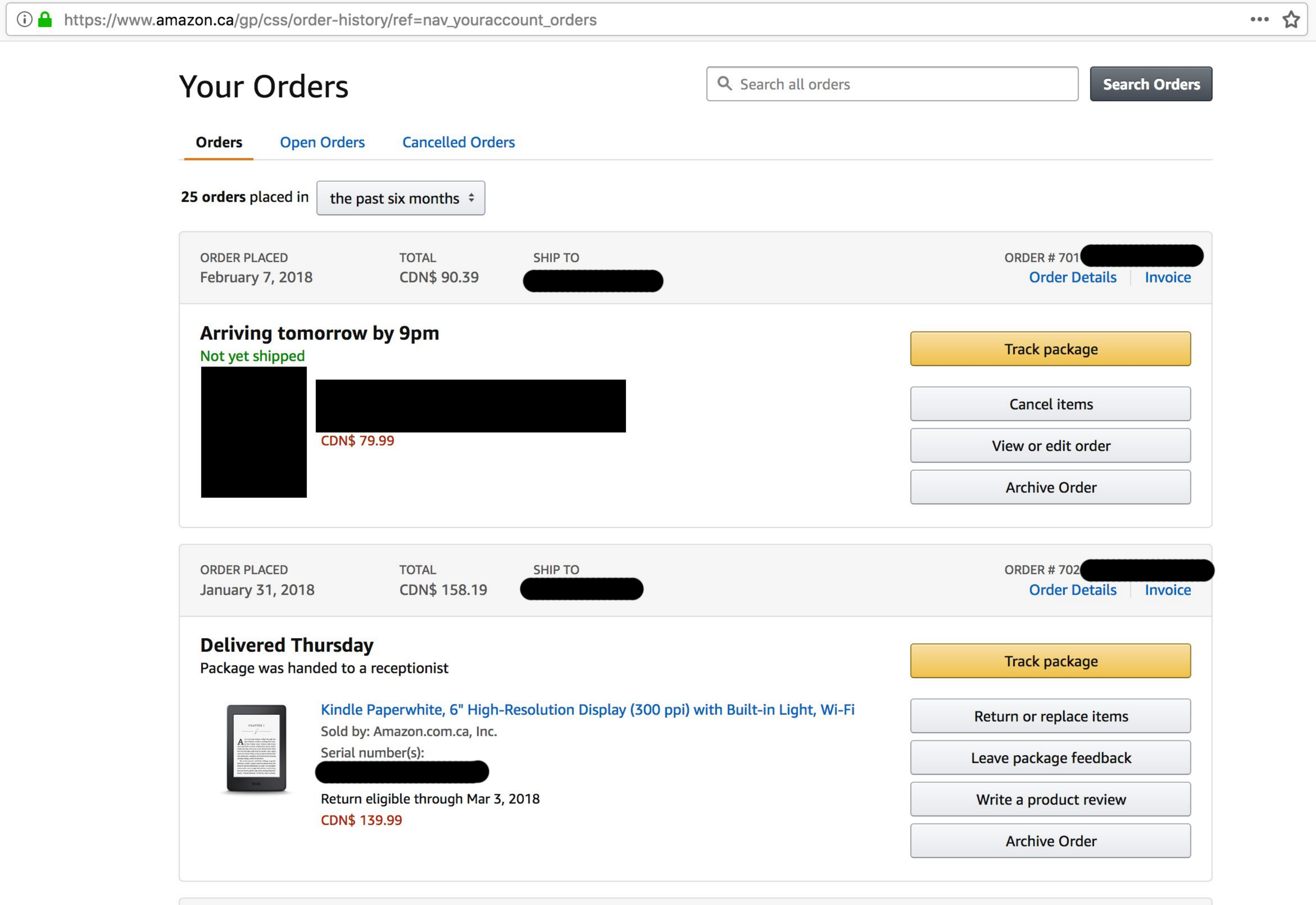
convert

spend

borrow

budget

balance



https://www.amazon.ca/gp/css/order-history/ref=nav_youraccount_orders

Your Orders

25 orders placed in the past six months

ORDER PLACED	TOTAL	SHIP TO	ORDER # 701[redacted]
February 7, 2018	CDN$ 90.39	[redacted]	Order Details | Invoice

Arriving tomorrow by 9pm

Not yet shipped

[redacted]

[redacted]

CDN$ 79.99

ORDER PLACED	TOTAL	SHIP TO	ORDER # 702[redacted]
January 31, 2018	CDN$ 158.19	[redacted]	Order Details | Invoice

Delivered Thursday

Package was handed to a receptionist

Kindle Paperwhite, 6" High-Resolution Display (300 ppi) with Built-in Light, Wi-Fi

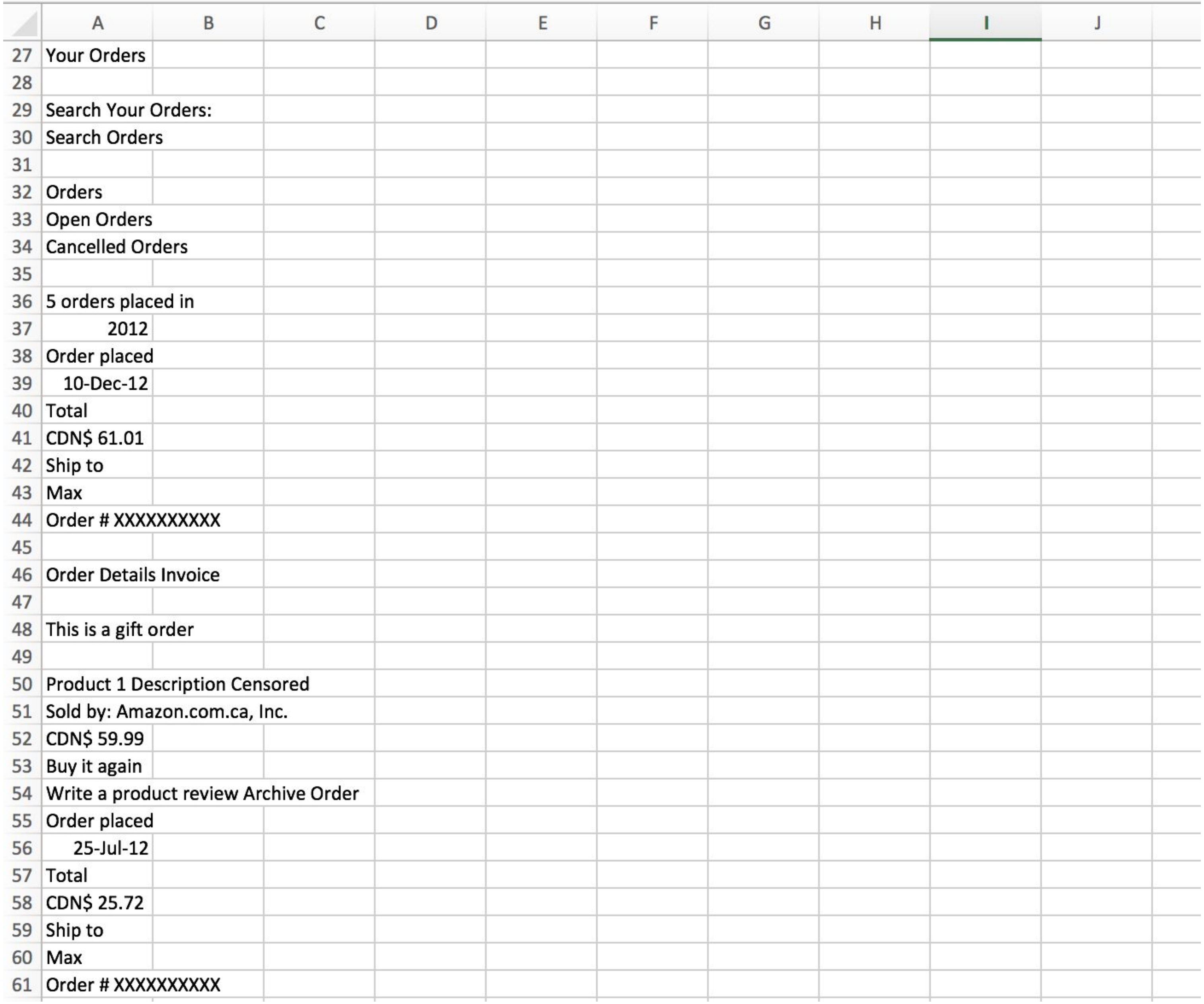
Sold by: Amazon.com.ca, Inc.

Serial number(s):

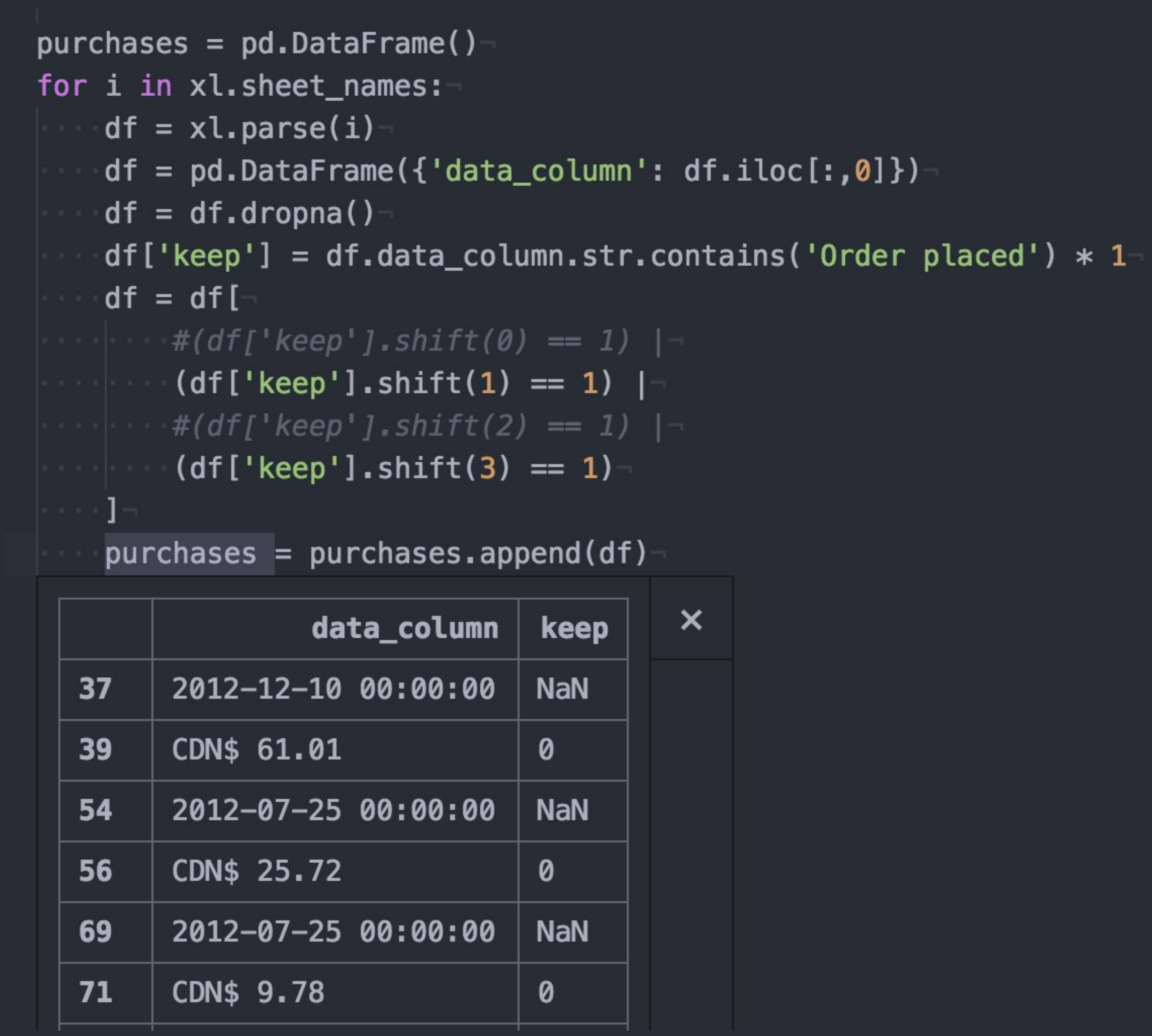
[redacted]

Return eligible through Mar 3, 2018

CDN$ 139.99

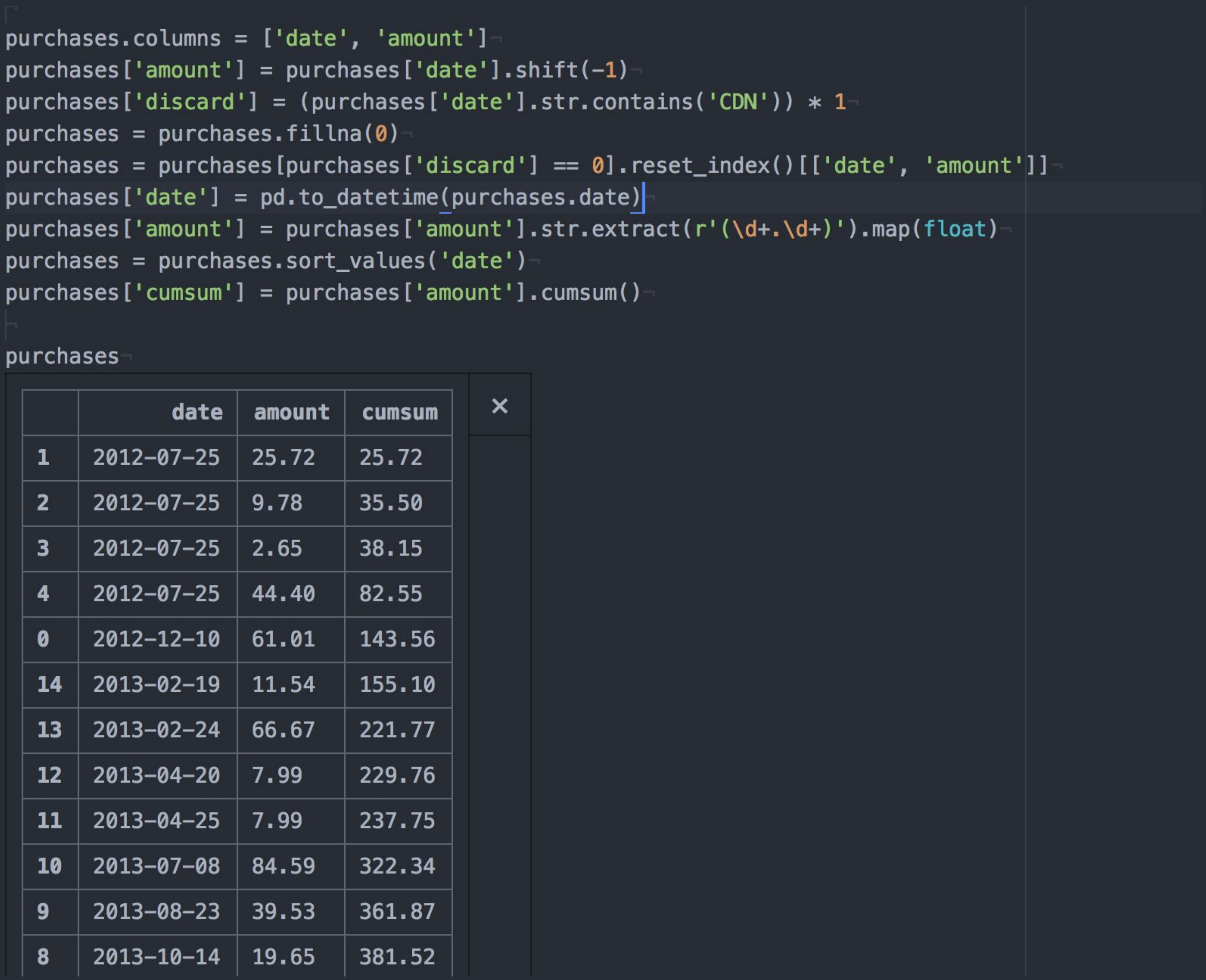


	A	B	C	D	E	F	G	H	I	J
27	Your Orders									
28										
29	Search Your Orders:									
30	Search Orders									
31										
32	Orders									
33	Open Orders									
34	Cancelled Orders									
35										
36	5 orders placed in									
37	2012									
38	Order placed									
39	10-Dec-12									
40	Total									
41	CDN$ 61.01									
42	Ship to									
43	Max									
44	Order # XXXXXXXXXX									
45										
46	Order Details Invoice									
47										
48	This is a gift order									
49										
50	Product 1 Description Censored									
51	Sold by: Amazon.com.ca, Inc.									
52	CDN$ 59.99									
53	Buy it again									
54	Write a product review Archive Order									
55	Order placed									
56	25-Jul-12									
57	Total									
58	CDN$ 25.72									
59	Ship to									
60	Max									
61	Order # XXXXXXXXXX									



```
purchases = pd.DataFrame()
for i in xl.sheet_names:
    df = xl.parse(i)
    df = pd.DataFrame({'data_column': df.iloc[:,0]})
    df = df.dropna()
    df['keep'] = df.data_column.str.contains('Order placed') * 1
    df = df[
        #(df['keep'].shift(0) == 1) |
        (df['keep'].shift(1) == 1) |
        #(df['keep'].shift(2) == 1) |
        (df['keep'].shift(3) == 1)
    ]
    purchases = purchases.append(df)
```

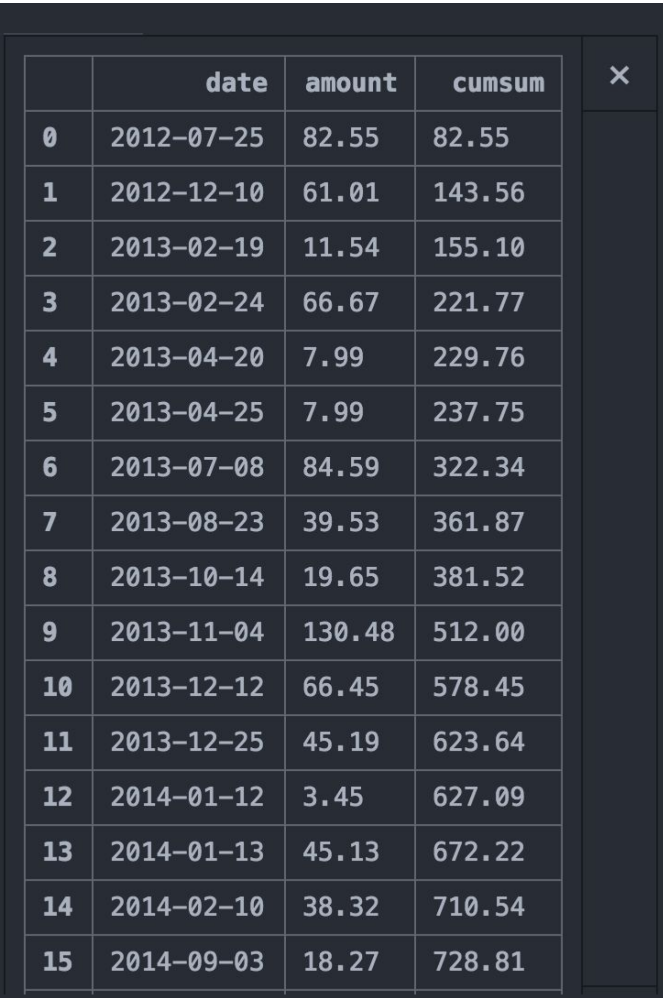
	data_column	keep
37	2012-12-10 00:00:00	NaN
39	CDN$ 61.01	0
54	2012-07-25 00:00:00	NaN
56	CDN$ 25.72	0
69	2012-07-25 00:00:00	NaN
71	CDN$ 9.78	0



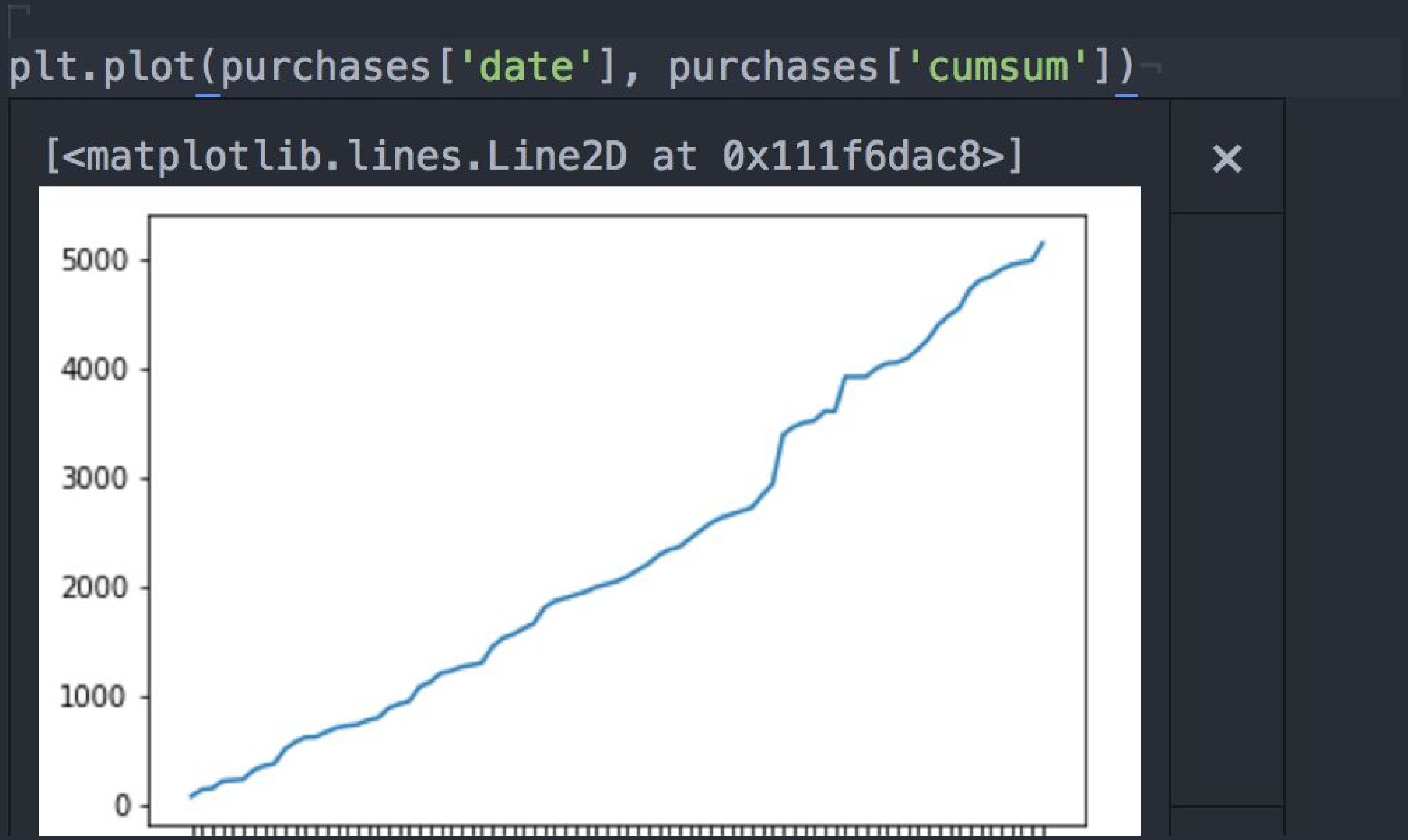
```
purchases.columns = ['date', 'amount']
purchases['amount'] = purchases['date'].shift(-1)
purchases['discard'] = (purchases['date'].str.contains('CDN')) * 1
purchases = purchases.fillna(0)
purchases = purchases[purchases['discard'] == 0].reset_index()[['date', 'amount']]
purchases['date'] = pd.to_datetime(purchases.date)
purchases['amount'] = purchases['amount'].str.extract(r'(\d+.\d+)').map(float)
purchases = purchases.sort_values('date')
purchases['cumsum'] = purchases['amount'].cumsum()

purchases
```

	date	amount	cumsum
1	2012-07-25	25.72	25.72
2	2012-07-25	9.78	35.50
3	2012-07-25	2.65	38.15
4	2012-07-25	44.40	82.55
0	2012-12-10	61.01	143.56
14	2013-02-19	11.54	155.10
13	2013-02-24	66.67	221.77
12	2013-04-20	7.99	229.76
11	2013-04-25	7.99	237.75
10	2013-07-08	84.59	322.34
9	2013-08-23	39.53	361.87
8	2013-10-14	19.65	381.52

```
purchases = pd.read_csv('data/purchases.csv')
purchases['cumsum'] = purchases['amount'].cumsum()
```



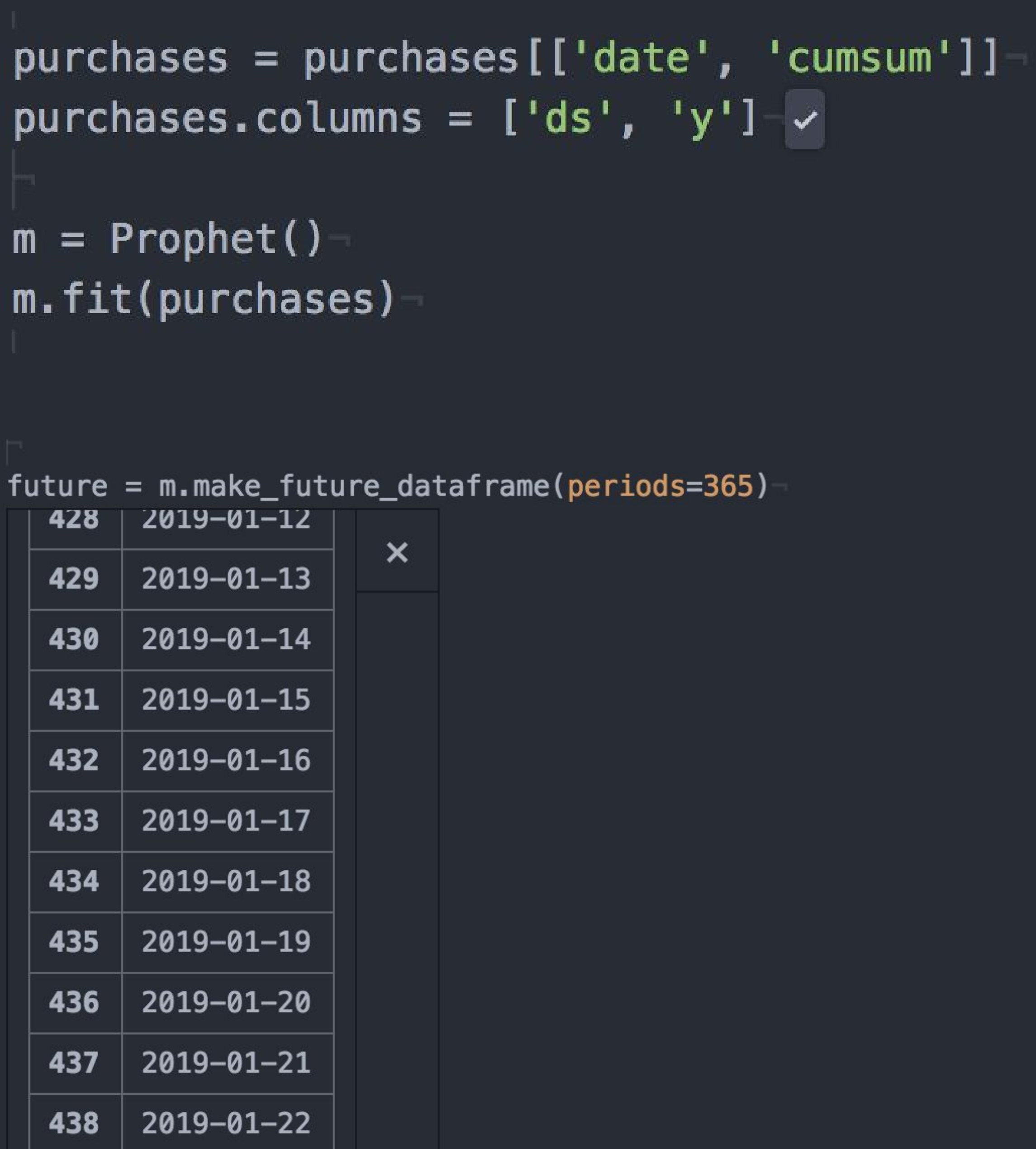
	date	amount	cumsum
0	2012-07-25	82.55	82.55
1	2012-12-10	61.01	143.56
2	2013-02-19	11.54	155.10
3	2013-02-24	66.67	221.77
4	2013-04-20	7.99	229.76
5	2013-04-25	7.99	237.75
6	2013-07-08	84.59	322.34
7	2013-08-23	39.53	361.87
8	2013-10-14	19.65	381.52
9	2013-11-04	130.48	512.00
10	2013-12-12	66.45	578.45
11	2013-12-25	45.19	623.64
12	2014-01-12	3.45	627.09
13	2014-01-13	45.13	672.22
14	2014-02-10	38.32	710.54
15	2014-09-03	18.27	728.81



```
plt.plot(purchases['date'], purchases['cumsum'])
```

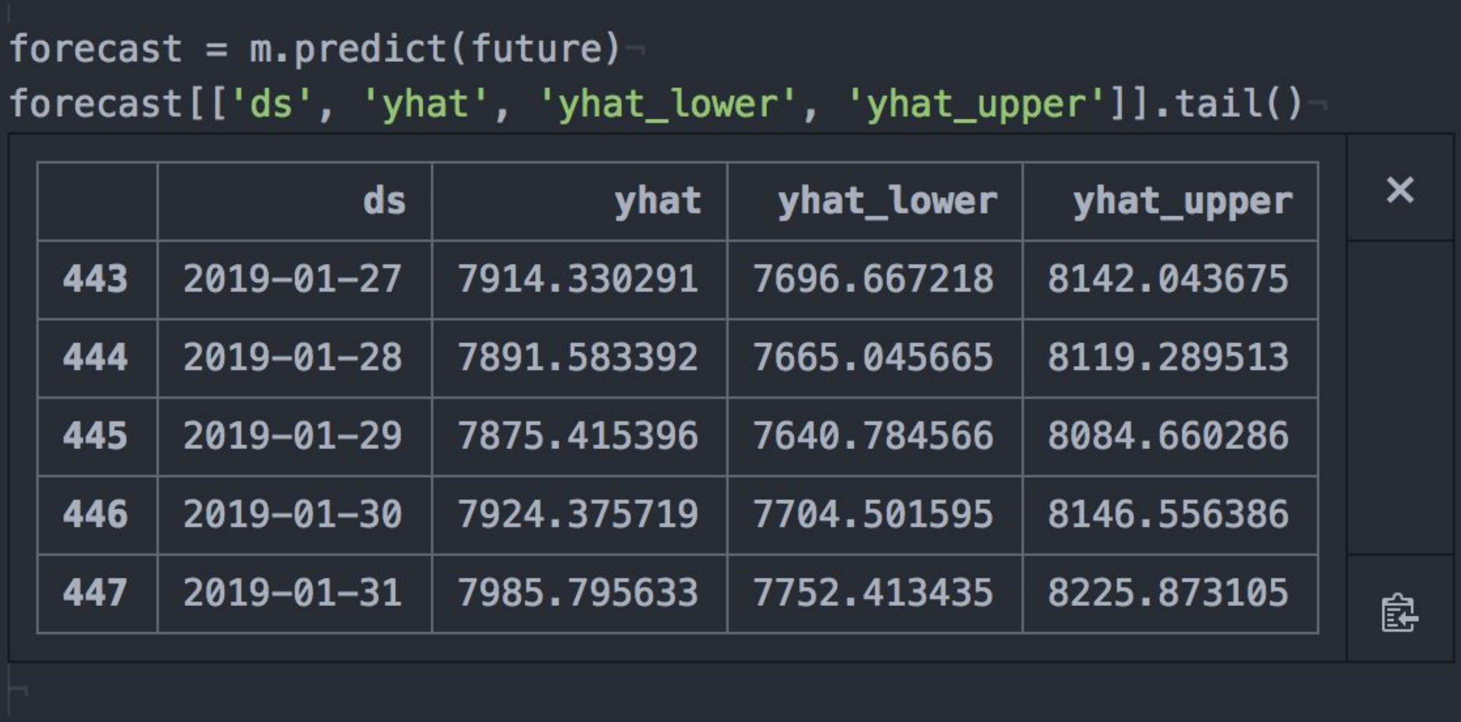
```
[<matplotlib.lines.Line2D at 0x111f6dac8>]
```

https://research.fb.com/prophet-forecasting-at-scale/



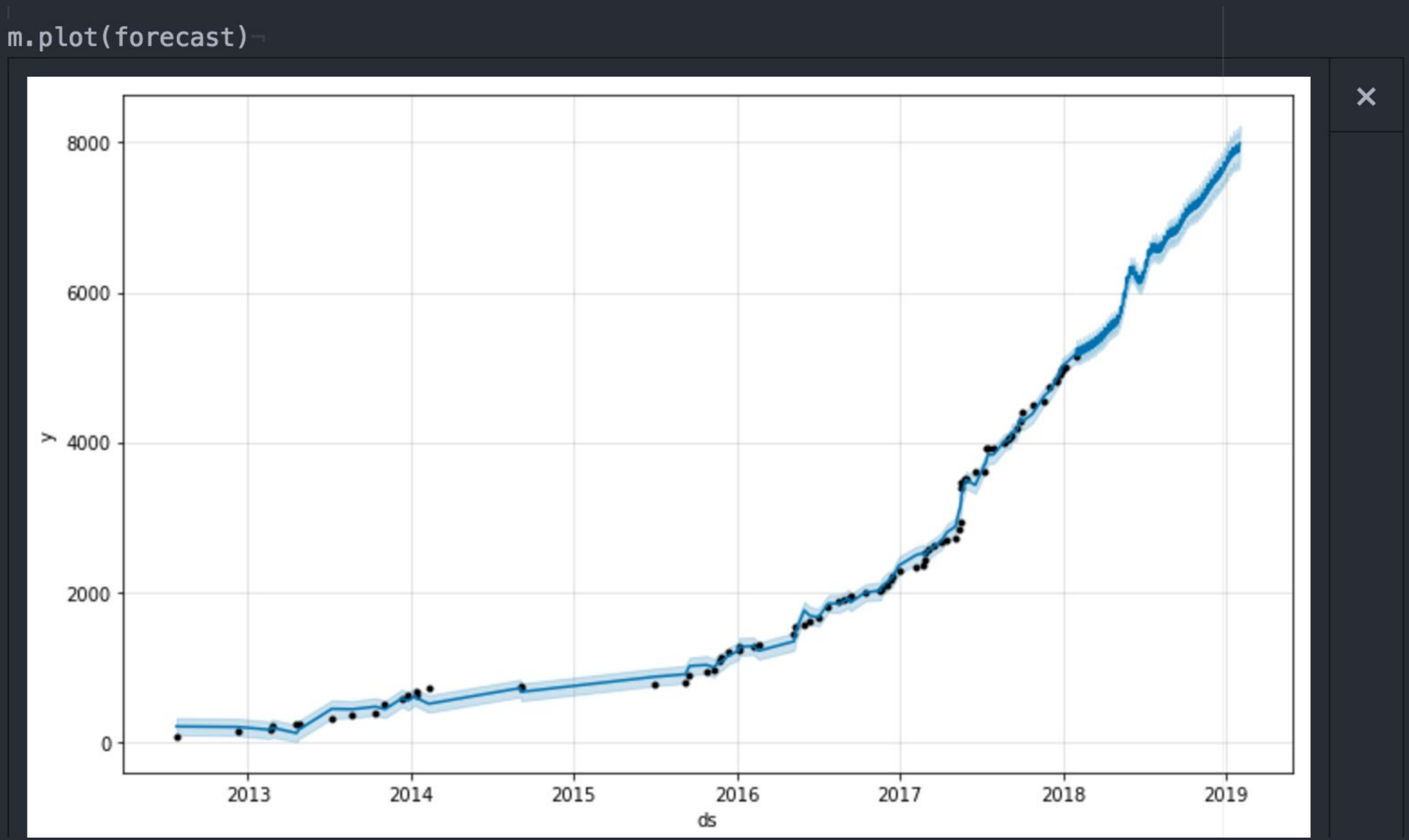
```python
purchases = purchases[['date', 'cumsum']]
purchases.columns = ['ds', 'y']

m = Prophet()
m.fit(purchases)


future = m.make_future_dataframe(periods=365)
```

428	2019-01-12
429	2019-01-13
430	2019-01-14
431	2019-01-15
432	2019-01-16
433	2019-01-17
434	2019-01-18
435	2019-01-19
436	2019-01-20
437	2019-01-21
438	2019-01-22

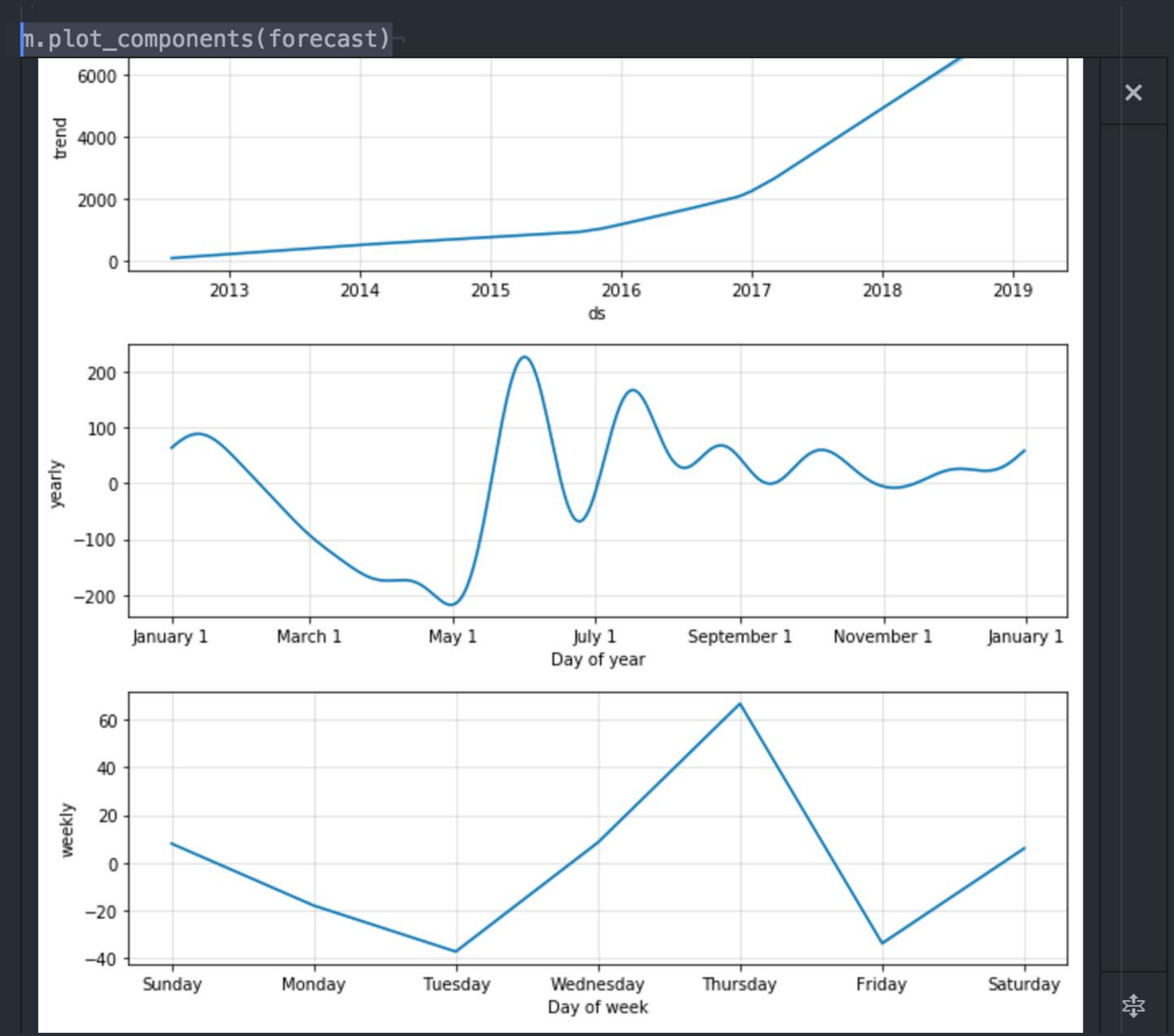


```
forecast = m.predict(future)
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

	ds	yhat	yhat_lower	yhat_upper
443	2019-01-27	7914.330291	7696.667218	8142.043675
444	2019-01-28	7891.583392	7665.045665	8119.289513
445	2019-01-29	7875.415396	7640.784566	8084.660286
446	2019-01-30	7924.375719	7704.501595	8146.556386
447	2019-01-31	7985.795633	7752.413435	8225.873105



```
m.plot(forecast)
```



```
m.plot_components(forecast)
```

FUN COUPONS!

thanks!

maxhumber

in