Building Better Budgets

🐦 @maxhumber

MinneFRAMA · 2018-12-03

https://github.com/maxhumber/bbb

“How much can I afford to sock away each month for my retirement, will my bank account survive a trip to Mauritius over Christmas, and should I quit my Bubble Tea habit?"

🐍 y Python tho?

Microsoft Excel

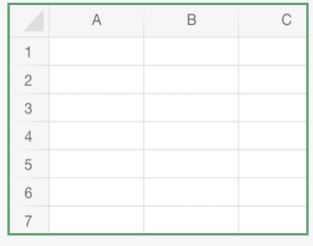
Search All Templates

Sign in

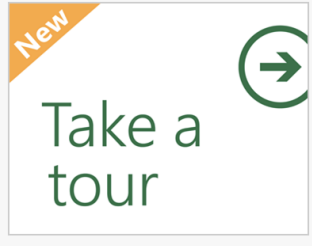
New

Recent

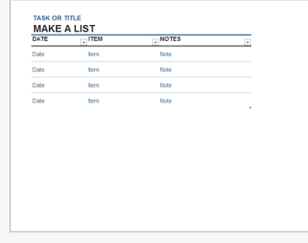
Open



Blank Workbook

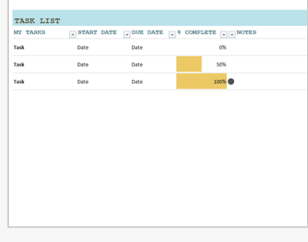


Welcome to Excel

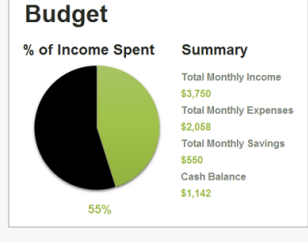


Make a List

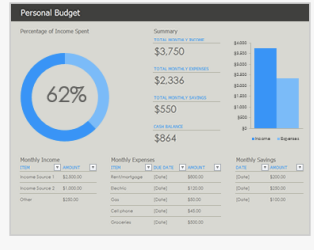
Total a list



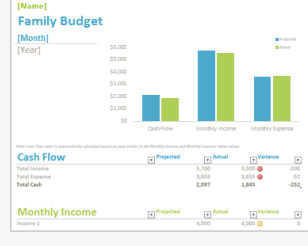
Track My Tasks



Manage My Money



Personal Budget



Family Budget

Cancel

Create

Budget

% of Income Spent

55%

Summary

Total Monthly Income	
$3,750	
Total Monthly Expenses	
$2,058	
Total Monthly Savings	
$550	
Cash Balance	
$1,142	

Monthly Income

Item	Amount
Income Source 1	$2,500.00
Income Source 2	$1,000.00
Other	$250.00

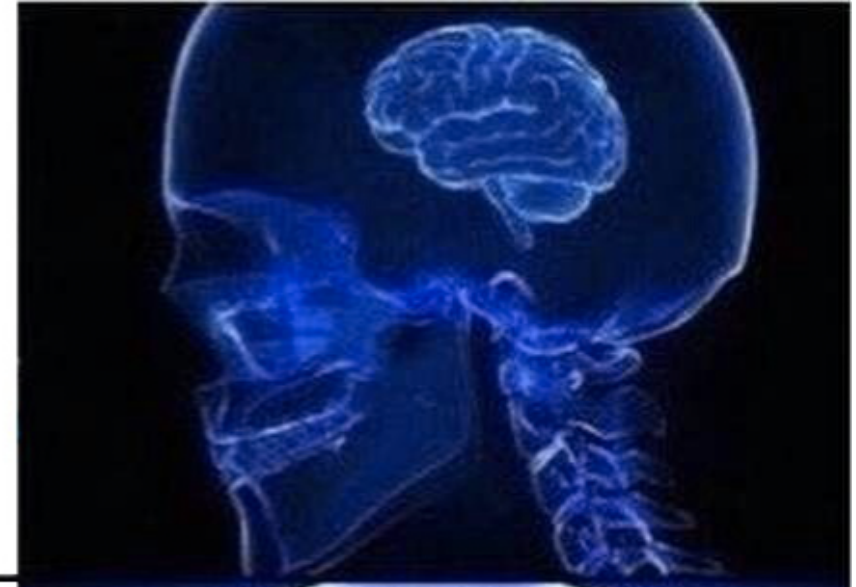
Monthly Expenses

Item	Amount
Rent/mortgage	$800.00
Electric	$120.00
Gas	$50.00
Cell phone	$45.00
Groceries	$500.00
Car payment	$273.00

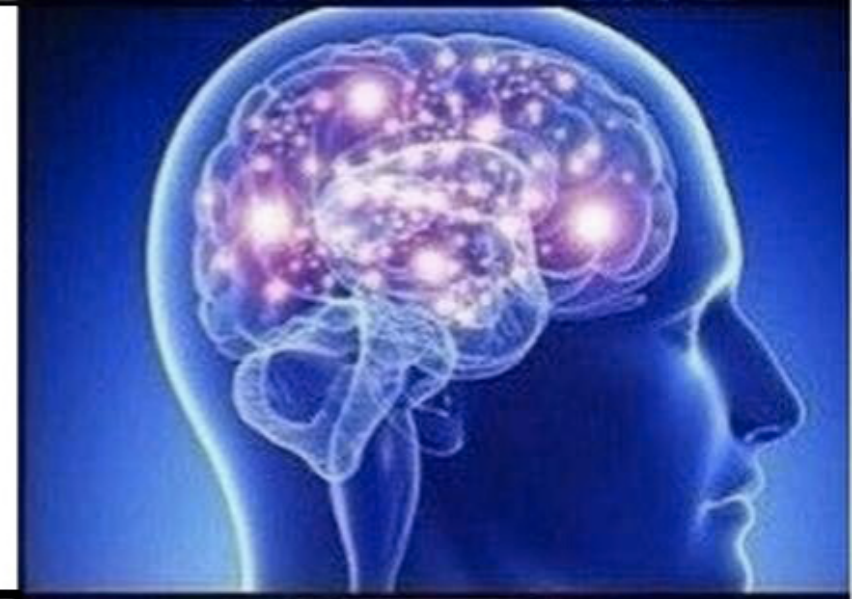
Bank	5000													
Visa	0													
Cash	0													
Months	4			4			4			4				
	WINTER 2015			SUMMER 2015			FALL 2015			WINTER 2016				
	One Time	Monthly	Total	One Time	Monthly	Total	One Time	Monthly	Total	One Time	Monthly	Total	Year Total	
FUNDING / INCOME														
Employment			0			0			0			0	0	
From Parents			0			0			0			0	0	
From Grants			0			0	900		900	900		900	1800	
From Financial Aid			0			0			0			0	0	
Transfer From Savings			0			0			0			0	0	
Other (Deposit)			0			0			0			0	0	
Total FUNDING / INCOME	0	0	0	0	0	0	900	0	900	900	0	900	1800	
EXPENSES														
Tuition			0			0	4100		4100	4100		4100	8200	
Books/Supplies			0			0	100		100	100		100	200	
Rent		600	2400		600	2400		500	2000		500	2000	8800	
Cell Phone		65	260		65	260		65	260		65	260	1040	
Laundry			0			0			0			0	0	
Grocery		300	1200		300	1200		300	1200		300	1200	4800	
Transportation		110	440		110	440			0			0	880	
Entertainment		300	1200		300	1200		300	1200		300	1200	4800	
Electronics			0			0			0			0	0	
Clothing			0			0			0			0	0	
Gifts			0			0			0			0	0	
Transfer to Savings			0			0			0			0	0	
Other			0			0			0			0	0	
SCENARIO PLANNING														
Mauritius			0			0	3000		3000			0	3000	
			0			0			0			0	0	
Total EXPENSES	0	1375	5500	0	1375	5500	7200	1165	11860	4200	1165	8860	31720	
NET			-5500			-5500			-10960			-7960	-29920	
Projected End			-500			-6000			-16960			-24920		

```
# DATE PARAMETERS
d1 <- Sys.Date()

# CREATE CALENDAR
df <- NULL
for (i in 1:365) {
    df <- rbind(df, (d1 - 1) + i)
}

# CREATE DATAFRAME
df <- as.data.frame(df)
names(df)[1] <- "date"

# ADD DAY IDENTIFIERS
df$date <- as.Date(df$date, origin = "1970-01-01")
df$month <- format(as.Date(df$date, format = "%Y-%m-%d"), "%B")
df$day <- format(as.Date(df$date, format = "%Y-%m-%d"), "%d")
df$weekday <- weekdays(as.Date(df$date))

# CREATE BUDGET
df$bank <- ifelse(df$date == d1, bank, 0)
df$income <- ifelse(
    (df$day == "15" | df$day == "30" | (df$day == "28" & df$month == "February")),
    round((salary*0.80/24),2), 0)
df$rent <- ifelse(df$day == "01" & df$date != "2017-04-01", -rent, 0)
df$internet <- ifelse(df$day == "01", -internet, 0)
df$hydro <- ifelse(df$day == "01", -hydro, 0)
df$phone <- ifelse(df$day == "25", -phone, 0)
df$grocery <- ifelse(df$weekday == "Sunday", -grocery, 0)
df$lunch <- ifelse(df$weekday != "Saturday" & df$weekday != "Sunday", -lunch, 0)
df$entertainment <- ifelse(df$weekday == "Friday" | df$weekday == "Saturday", -(entertainment/2), 0)
df$savings <- ifelse(df$weekday == "Monday", -savings, 0)
df$osap <- ifelse(df$weekday == "Monday", -osap, 0)
df$other <- ifelse(df$weekday == "Monday", -other, 0)
```

Excel



Custom

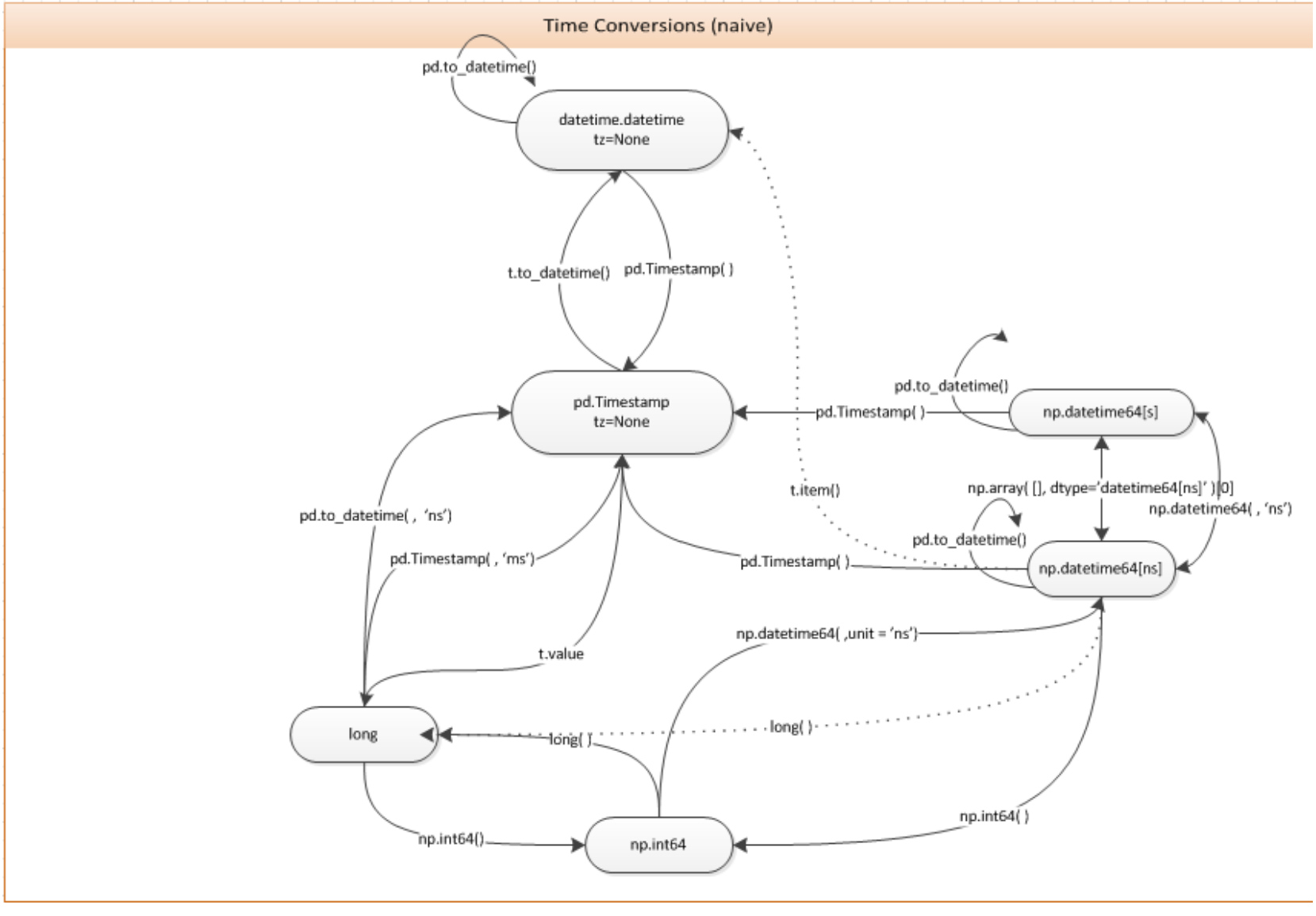


The actual reason 📦

- `Altair`
- `Recurrent`
- `yaml`
- `Fire`

Max's Rules of Dating

Dates are a mess (https://i.stack.imgur.com/uiXQd.png)


Time Conversions (naive)
pd.to_datetime()
datetime.datetime
tz=None
t.to_datetime()
pd.Timestamp( )
pd.Timestamp
tz=None
pd.to_datetime()
pd.Timestamp( )
np.datetime64[s]
t.item()
np.array([], dtype='datetime64[ns]')[0]
np.datetime64(, 'ns')
pd.to_datetime(, 'ns')
pd.to_datetime()
pd.Timestamp(, 'ms')
pd.Timestamp( )
np.datetime64[ns]
np.datetime64(,unit = 'ns')
t.value
long
long( )
long( )
np.int64()
np.int64
np.int64( )

Max's Rules of Dating

1. Use `Timestamp`
2. `normalize` all the things

STFU AND SHOW ME SOME CODE

In [1]:
```
import datetime
from dateutil import rrule

import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline

from recurrent import RecurringEvent  # pip install recurrent
import yaml  # pip install pyyaml
import altair as alt
```

In [2]:
```
datetime.datetime.now()
# datetime.date.today()
```

Out[2]:
```
datetime.datetime(2018, 12, 3, 9, 34, 52, 446050)
```

In [3]:
```
datetime.datetime(1993, 6, 7, 15, 16, 0)
```

Out[3]:
```
datetime.datetime(1993, 6, 7, 15, 16)
```

Timestamp

In [4]:
```
date_1 = datetime.datetime.now()

print(pd.Timestamp(date_1))
print(pd.to_datetime(date_1))
```

```
2018-12-03 09:34:52.470373
2018-12-03 09:34:52.470373
```

In [5]:
```
date_2 = pd.Timestamp(1993, 6, 7, 15, 16, 0)
date_2
```

Out[5]:
```
Timestamp('1993-06-07 15:16:00')
```

In [6]:
```
date_3 = pd.Timestamp('2018-12-03')
date_3
```

Out[6]:
```
Timestamp('2018-12-03 00:00:00')
```

In [7]:
```
type(date_3)
```

Out[7]:
```
pandas._libs.tslibs.timestamps.Timestamp
```

In [8]:
```
[method for method in dir(date_3) if '__' not in method][10:20]
```

Out[8]:
```
['astimezone',
 'ceil',
 'combine',
 'ctime',
 'date',
 'day',
 'day_name',
 'dayofweek',
 'dayofyear',
 'days_in_month']
```

.normalize

In [9]:
```
print(date_1)
date_1 = pd.Timestamp(date_1)
print(date_1)
print(date_1.normalize())
```

```
2018-12-03 09:34:52.470373
2018-12-03 09:34:52.470373
2018-12-03 00:00:00
```

Calendar

In [10]:
```
start = pd.Timestamp('now').normalize()
end = start + pd.Timedelta('365 days')
```

In [11]:
```
calendar = pd.DataFrame(index=pd.date_range(start, end))
```

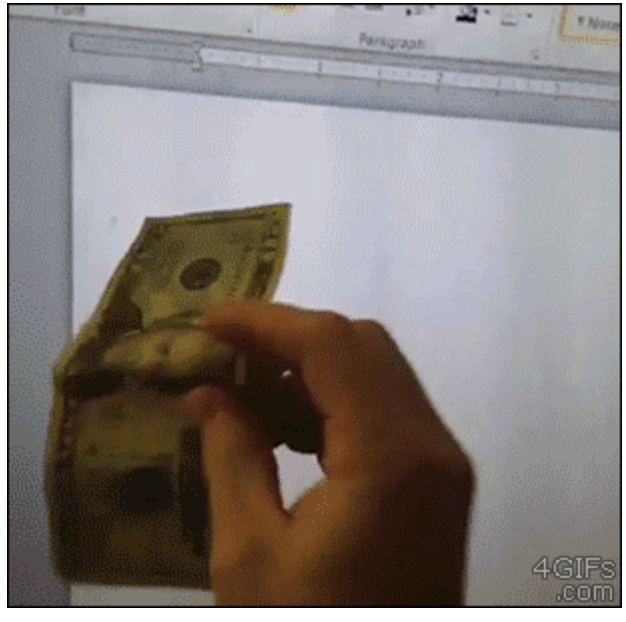
In [12]:
```
calendar.head()
```

Out[12]:

2018-12-03
2018-12-04
2018-12-05
2018-12-06
2018-12-07

Hydrate

- Income: twice a month ($1000)
- Rent: once a month ($1500)


Paragraph
4GIFs
.com

Offset Aliases

A number of string aliases are given to useful common time series frequencies. We will refer to these aliases as *offset aliases*.

Alias	Description
B	business day frequency
C	custom business day frequency
D	calendar day frequency
W	weekly frequency
M	month end frequency
SM	semi-month end frequency (15th and end of month)
BM	business month end frequency
CBM	custom business month end frequency
MS	month start frequency
SMS	semi-month start frequency (1st and 15th)
BMS	business month start frequency
CBMS	custom business month start frequency
Q	quarter end frequency
BQ	business quarter end frequency
QS	quarter start frequency
BQS	business quarter start frequency
A, Y	year end frequency
BA, BY	business year end frequency
AS, YS	year start frequency
BAS, BYS	business year start frequency
BH	business hour frequency
H	hourly frequency
T, min	minutely frequency
S	secondly frequency
L, ms	milliseconds
U, us	microseconds
N	nanoseconds

In [13]:
```
income = pd.DataFrame(
    data={'income': 1000},
    index=pd.date_range(start, end, freq='SM')
)
income.head()
```

Out[13]:

	income
2018-12-15	1000
2018-12-31	1000
2019-01-15	1000
2019-01-31	1000
2019-02-15	1000

In [14]:
```
rent = pd.DataFrame(
    data={'rent': -1500},
    index=pd.date_range(start, end, freq='MS')
)
rent.head()
```

Out[14]:

	rent
2019-01-01	-1500
2019-02-01	-1500
2019-03-01	-1500
2019-04-01	-1500
2019-05-01	-1500

In [15]:
```
calendar = pd.concat([calendar, income], axis=1).fillna(0)
calendar = pd.concat([calendar, rent], axis=1).fillna(0)
calendar.iloc[5:15]
```

Out[15]:

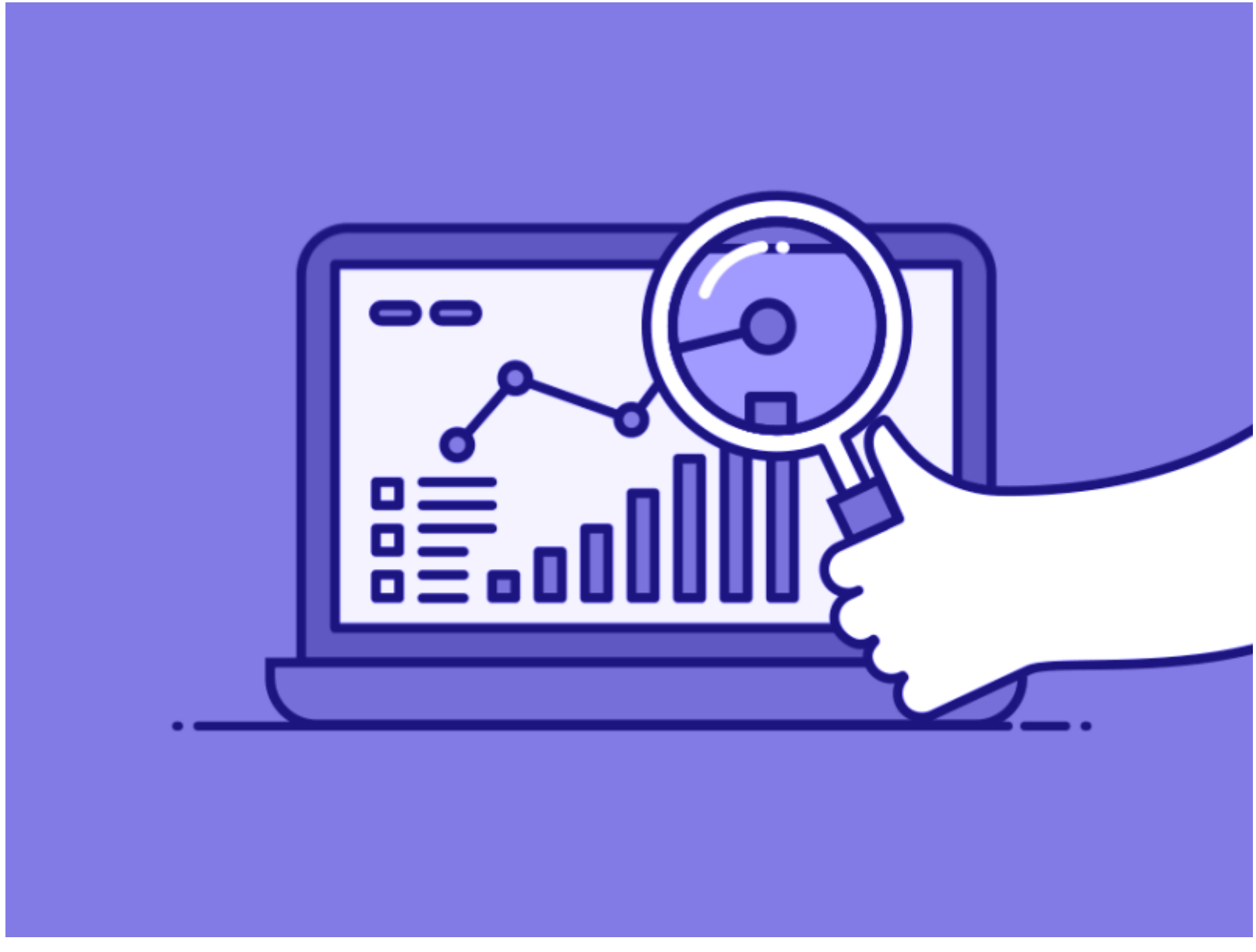
	income	rent
2018-12-08	0.0	0.0
2018-12-09	0.0	0.0
2018-12-10	0.0	0.0
2018-12-11	0.0	0.0
2018-12-12	0.0	0.0
2018-12-13	0.0	0.0
2018-12-14	0.0	0.0
2018-12-15	1000.0	0.0
2018-12-16	0.0	0.0
2018-12-17	0.0	0.0

In [16]:
```
calendar['total'] = calendar.sum(axis=1)
calendar['cum_total'] = calendar['total'].cumsum()
```

In [17]:
```
calendar.tail(1)
```

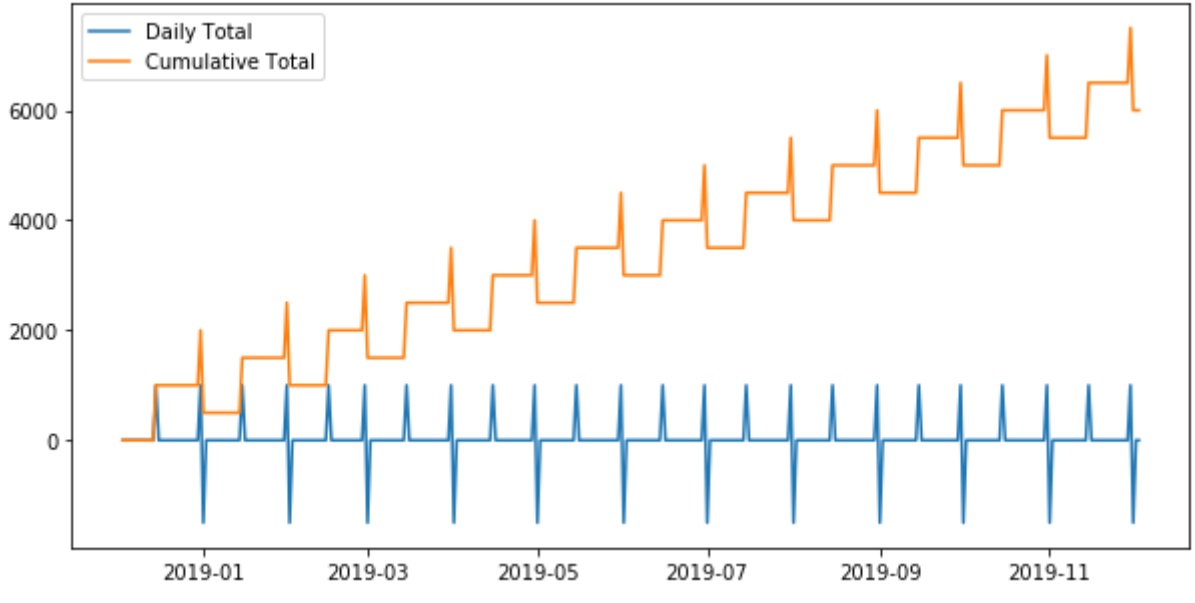
Out[17]:

	income	rent	total	cum_total
2019-12-03	0.0	0.0	0.0	6000.0



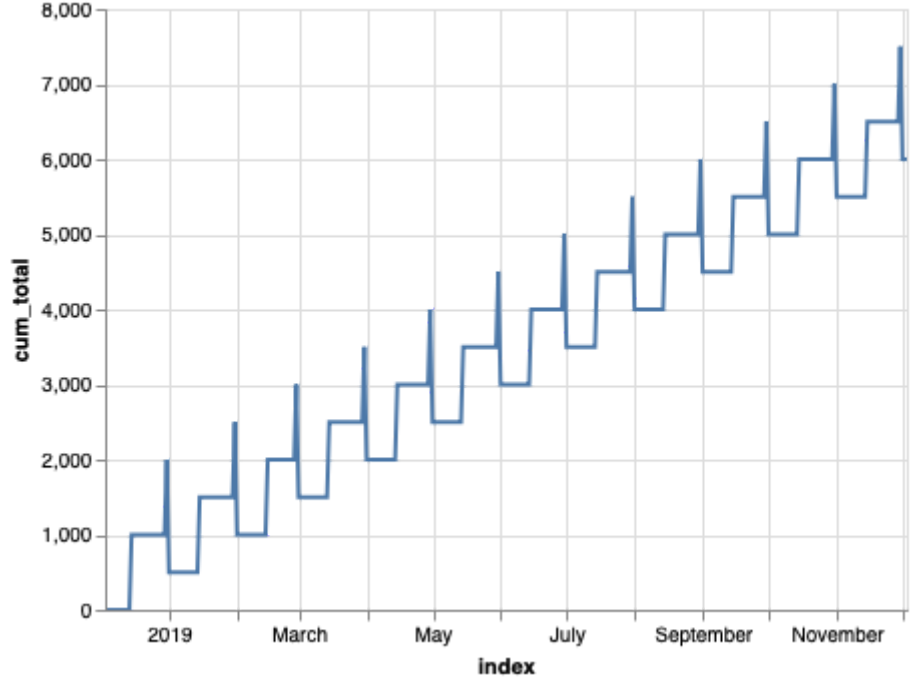
In [18]:
```
plt.figure(figsize=(10, 5))
plt.plot(calendar.index, calendar.total, label='Daily Total')
plt.plot(calendar.index, calendar.cum_total, label='Cumulative Total')
plt.legend()
```

Out[18]:
```
<matplotlib.legend.Legend at 0x11b5379b0>
```



In [19]:
```
import altair as alt
alt.renderers.enable('notebook')

alt.Chart(calendar.reset_index())\
    .mark_line()\
    .encode(x='index', y='cum_total')
```

Out[19]:

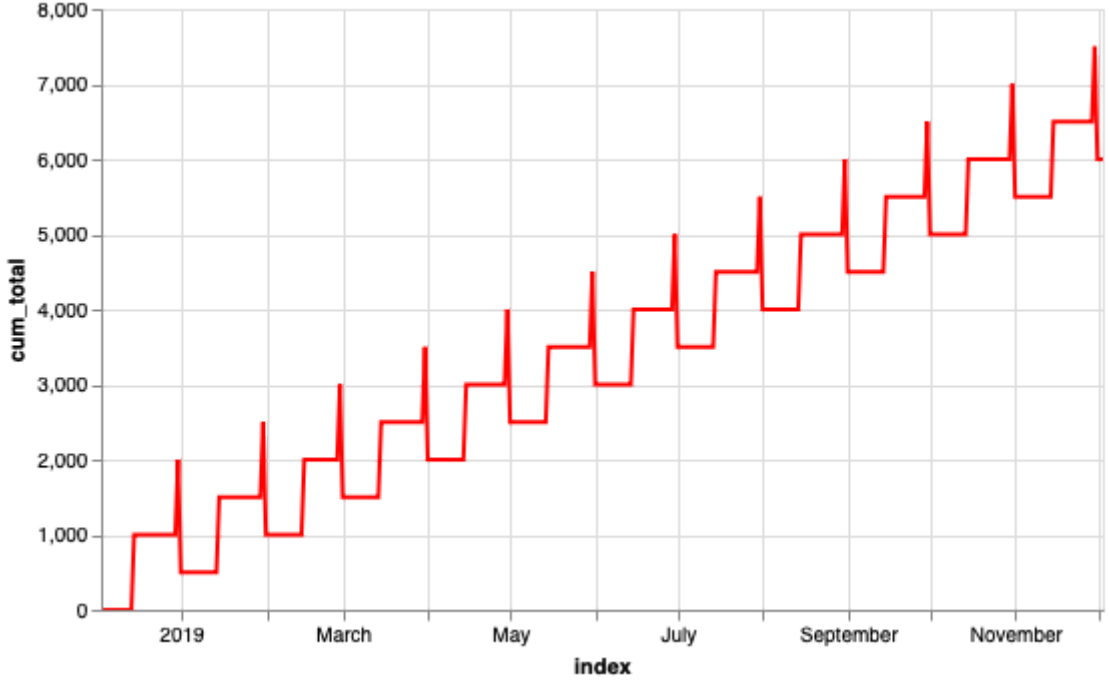


In [20]:

```
vis = (
    alt.Chart(calendar.reset_index())
    .mark_line(color='red')
    .encode(
        x='index',
        y='cum_total',
        tooltip='cum_total'
    )
    .interactive()
    .properties(width=500, height=300)
)
```

In [21]:
```
vis
```

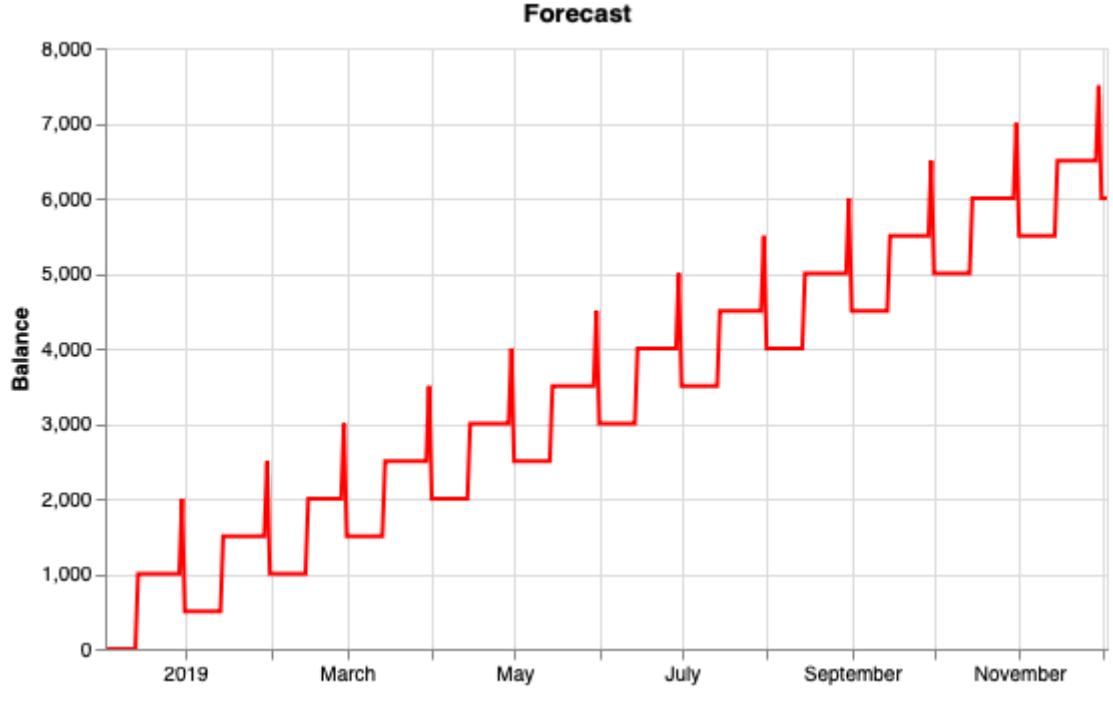
Out[21]:



In [22]:

```
def plot_budget(calendar):
    vis = (
        alt.Chart(calendar.reset_index())
        .mark_line(color='red')
        .encode(
            x=alt.X('index', title=''),
            y=alt.Y('cum_total', title='Balance'),
            tooltip='cum_total'
        )
        .interactive()
        .properties(width=500, height=300, title='Forecast')
    )
    vis.display()
```

In [23]:
```
plot_budget(calendar)
```



In [24]:
```
bank = pd.DataFrame(
    data={'bank': 8000},
    index=pd.date_range(start, end=start)
)
print(bank)
```

```
            bank
2018-12-03  8000
```

In [25]:
```
calendar = pd.concat([calendar, bank], axis=1).fillna(0)
```

In [26]:
```
calendar.sum(axis=1).head()
```

Out[26]:
```
2018-12-03    8000.0
2018-12-04       0.0
2018-12-05       0.0
2018-12-06       0.0
2018-12-07       0.0
Freq: D, dtype: float64
```

In [27]:
```
def update_totals(df):
    # check to see if these columns exit in our dataframe
    if df.columns.isin(['total', 'cum_total']).any():
        # if they do exist set the them to 0
        df['total'] = 0
        df['cum_total'] = 0
    # recalculate total and cumulative_total
    df['total'] = df.sum(axis=1)
    df['cum_total'] = df['total'].cumsum()
    return df
```

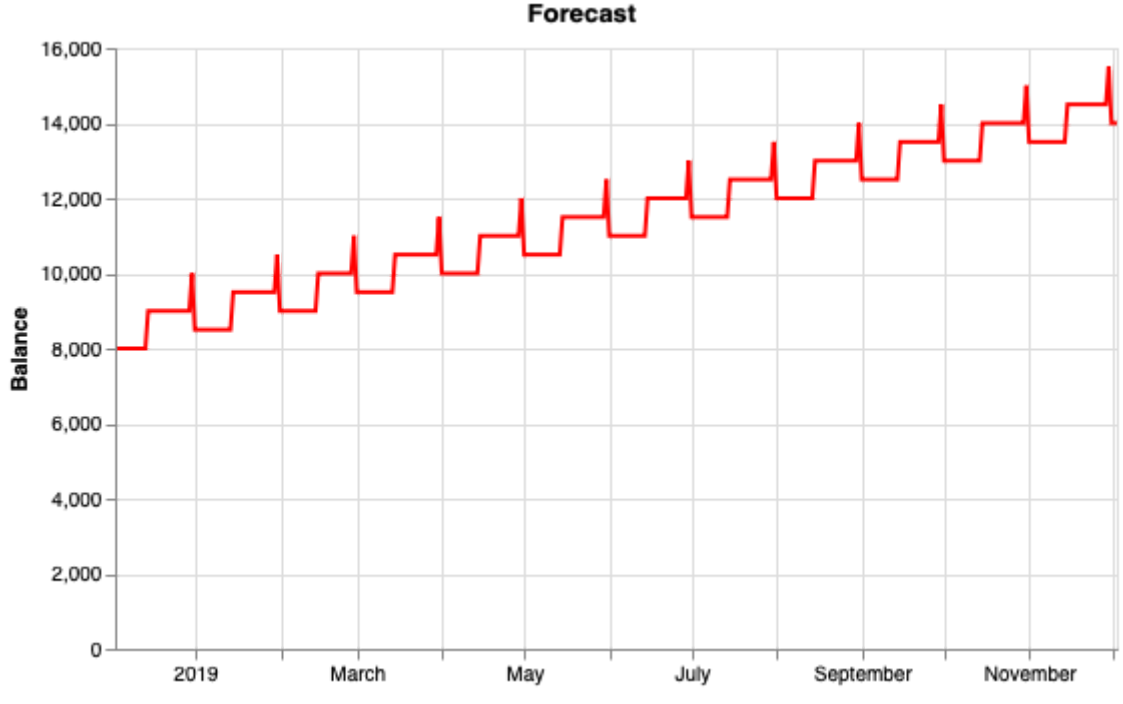
In [28]:
```
calendar = update_totals(calendar)
```

In [29]:
```
calendar.tail(5)
```

Out[29]:

	income	rent	total	cum_total	bank
2019-11-29	0.0	0.0	0.0	14500.0	0.0
2019-11-30	1000.0	0.0	1000.0	15500.0	0.0
2019-12-01	0.0	-1500.0	-1500.0	14000.0	0.0
2019-12-02	0.0	0.0	0.0	14000.0	0.0
2019-12-03	0.0	0.0	0.0	14000.0	0.0

In [30]:
```
plot_budget(calendar)
```



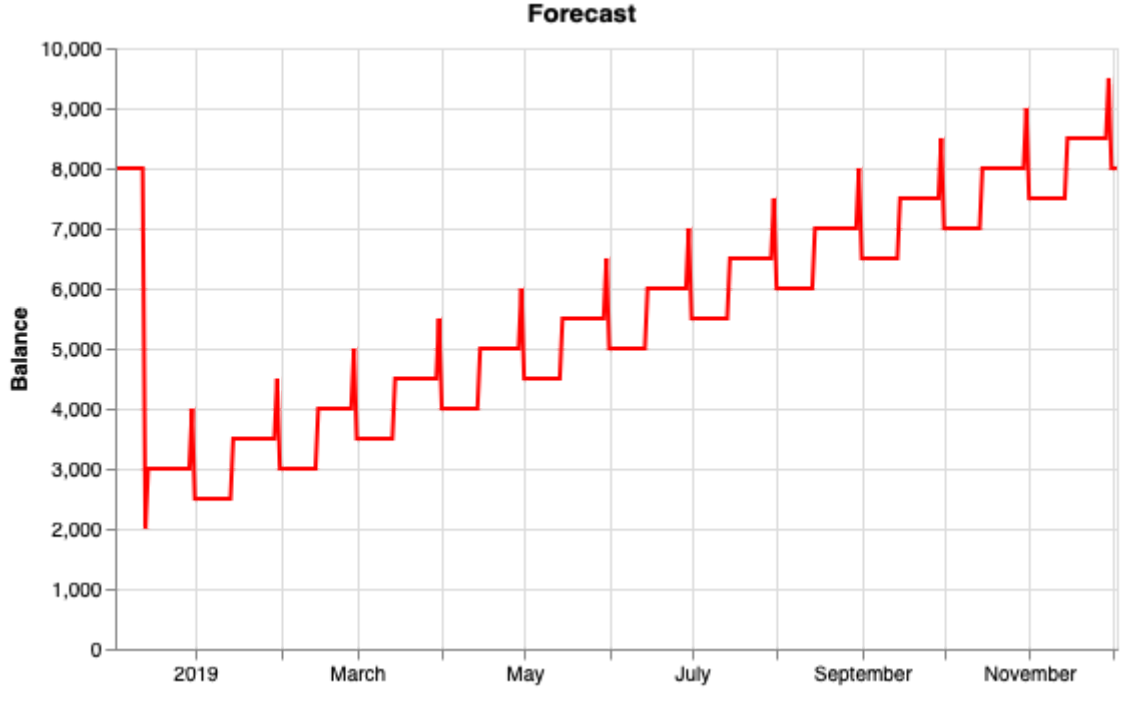
In [31]:
```
vacation = pd.DataFrame(
    data={'vacation': -6000},
    index=[pd.Timestamp('2018-12-14').normalize()]
)
print(vacation)
```

```
            vacation
2018-12-14     -6000
```

In [32]:
```
calendar = pd.concat([calendar, vacation], axis=1).fillna(0)
```

In [33]:
```
calendar = update_totals(calendar)
```

In [34]:
```
plot_budget(calendar)
```



More...

In [35]:
```
start = pd.Timestamp('now').normalize()
end = start + pd.Timedelta('365 days')
```

In [36]:
```
# every day
pd.date_range(start, end, freq='D')[:10]
```

Out[36]:
```
DatetimeIndex(['2018-12-03', '2018-12-04', '2018-12-05', '2018-12-06',
               '2018-12-07', '2018-12-08', '2018-12-09', '2018-12-10',
               '2018-12-11', '2018-12-12'],
              dtype='datetime64[ns]', freq='D')
```

In [37]:
```
# every weekday
pd.date_range(start, end, freq='B')[:10]
```

Out[37]:
```
DatetimeIndex(['2018-12-03', '2018-12-04', '2018-12-05', '2018-12-06',
               '2018-12-07', '2018-12-10', '2018-12-11', '2018-12-12',
               '2018-12-13', '2018-12-14'],
              dtype='datetime64[ns]', freq='B')
```

In [38]:
```
# every weekend
pd.DatetimeIndex(
    set(pd.date_range(start, end, freq='D')) - set(pd.date_range(start, end, freq=
'B'))
)[:10]
```

Out[38]:
```
DatetimeIndex(['2019-06-29', '2019-06-15', '2019-04-14', '2019-11-23',
               '2019-11-02', '2019-03-17', '2019-01-06', '2019-09-07',
               '2019-04-13', '2019-10-19'],
              dtype='datetime64[ns]', freq=None)
```

In [39]:
```
# every week
# pd.date_range(start, end, freq='W') # bad
# start.day_name()
pd.date_range(start, end, freq='7D')[:10]
```

Out[39]:
```
DatetimeIndex(['2018-12-03', '2018-12-10', '2018-12-17', '2018-12-24',
               '2018-12-31', '2019-01-07', '2019-01-14', '2019-01-21',
               '2019-01-28', '2019-02-04'],
              dtype='datetime64[ns]', freq='7D')
```

In [40]:
```
# every two weeks
# pd.date_range(start, end, freq='2W') # bad
pd.date_range(start, end, freq='14D')[:10]
```

Out[40]:
```
DatetimeIndex(['2018-12-03', '2018-12-17', '2018-12-31', '2019-01-14',
               '2019-01-28', '2019-02-11', '2019-02-25', '2019-03-11',
               '2019-03-25', '2019-04-08'],
              dtype='datetime64[ns]', freq='14D')
```

In [41]:
```
# every month
pd.date_range(start, end, freq='M').shift(7, freq='D')[:10]
```

Out[41]:
```
DatetimeIndex(['2019-01-07', '2019-02-07', '2019-03-07', '2019-04-07',
               '2019-05-07', '2019-06-07', '2019-07-07', '2019-08-07',
               '2019-09-07', '2019-10-07'],
              dtype='datetime64[ns]', freq=None)
```

In [42]:
```
# SemiMonthEnd 15th and calendar month end
pd.date_range(start, end, freq='SM')[:10]
```

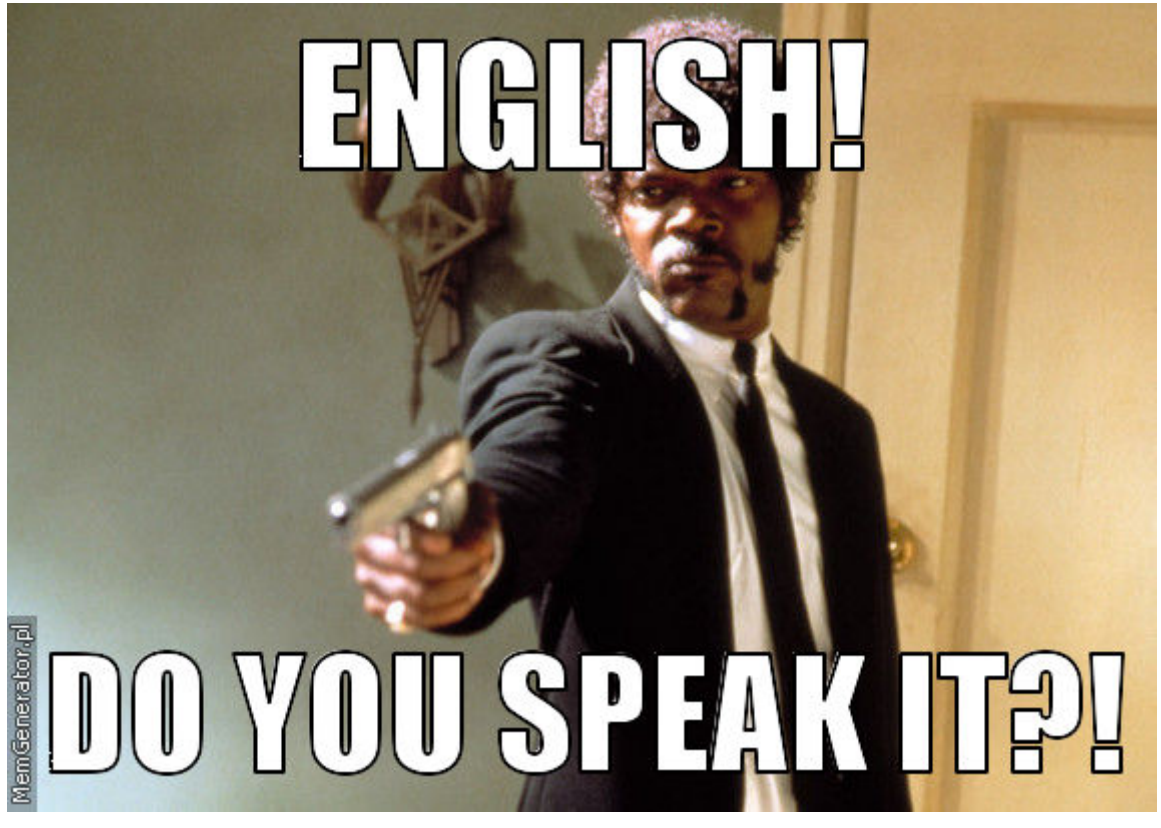
Out[42]:
```
DatetimeIndex(['2018-12-15', '2018-12-31', '2019-01-15', '2019-01-31',
               '2019-02-15', '2019-02-28', '2019-03-15', '2019-03-31',
               '2019-04-15', '2019-04-30'],
              dtype='datetime64[ns]', freq='SM-15')
```

In [43]:

```
anchor = '2018-12-03'
frequency = 'every week'

def generate_dates(start, end, anchor, frequency):
    if frequency == 'every day':
        return pd.date_range(start, end, freq='D')
    if frequency == 'every weekday':
        return pd.date_range(start, end, freq='B')
    if frequency == 'every weekend':
        return pd.DatetimeIndex(
            set(pd.date_range(start, end, freq='D')) -
            set(pd.date_range(start, end, freq='B'))
        )
    if frequency == 'every week':
        return pd.date_range(anchor, end, freq='7D')
    if frequency in ('every two weeks', 'every other week'):
        return pd.date_range(anchor, end, freq='14D')
    if frequency == 'every month':
        d = pd.Timestamp(anchor).day
        return pd.date_range(start, end, freq='M').shift(d, freq='D')
    if frequency == 'twice a month':
        return pd.date_range(start, end, freq='SM')
```

In [44]:
```
generate_dates(start, end, '2018-12-03', 'every week')[:10]
```

Out[44]:
```
DatetimeIndex(['2018-12-03', '2018-12-10', '2018-12-17', '2018-12-24',
               '2018-12-31', '2019-01-07', '2019-01-14', '2019-01-21',
               '2019-01-28', '2019-02-04'],
              dtype='datetime64[ns]', freq='7D')
```


ENGLISH!
DO YOU SPEAK IT?!
MemGenerator.pl

In [45]:
```
frequency = 'every week until July 10th'
```

```
!pip install recurrent
```

In [46]:
```
from recurrent import RecurringEvent
```

In [47]:
```
r = RecurringEvent()
r.parse(frequency)

# will allow us to generate a recurrance rule (rrule) that is iCalendar RFC compli
ant.
```

Out[47]:
```
'RRULE:INTERVAL=1;FREQ=WEEKLY;UNTIL=20190710'
```

In [48]:
```
from dateutil import rrule
```

In [49]:
```
rr = rrule.rrulestr(r.get_RFC_rrule())
rr
```

Out[49]:
```
<dateutil.rrule.rrule at 0x109e18438>
```

In [50]:
```
rr.between(start, end)[:10]
```

Out[50]:
```
[datetime.datetime(2018, 12, 3, 9, 34, 53),
 datetime.datetime(2018, 12, 10, 9, 34, 53),
 datetime.datetime(2018, 12, 17, 9, 34, 53),
 datetime.datetime(2018, 12, 24, 9, 34, 53),
 datetime.datetime(2018, 12, 31, 9, 34, 53),
 datetime.datetime(2019, 1, 7, 9, 34, 53),
 datetime.datetime(2019, 1, 14, 9, 34, 53),
 datetime.datetime(2019, 1, 21, 9, 34, 53),
 datetime.datetime(2019, 1, 28, 9, 34, 53),
 datetime.datetime(2019, 2, 4, 9, 34, 53)]
```

In [51]:
```
[pd.to_datetime(date).normalize() for date in rr.between(start, end)][:10]
```

Out[51]:
```
[Timestamp('2018-12-03 00:00:00'),
 Timestamp('2018-12-10 00:00:00'),
 Timestamp('2018-12-17 00:00:00'),
 Timestamp('2018-12-24 00:00:00'),
 Timestamp('2018-12-31 00:00:00'),
 Timestamp('2019-01-07 00:00:00'),
 Timestamp('2019-01-14 00:00:00'),
 Timestamp('2019-01-21 00:00:00'),
 Timestamp('2019-01-28 00:00:00'),
 Timestamp('2019-02-04 00:00:00')]
```

In [52]:

```
def get_dates(frequency):
    # let pandas try to do it's thing
    try:
        return [pd.Timestamp(frequency).normalize()]
    except ValueError:
        pass
    # parse frequency with recurrent
    try:
        r = RecurringEvent()
        r.parse(frequency)
        rr = rrule.rrulestr(r.get_RFC_rrule())
        return [
            pd.to_datetime(date).normalize()
            for date in rr.between(start, end)
        ]
    except ValueError:
        raise ValueError('Invalid frequency')
```

In [53]:
```
get_dates('2019-01-01')
```

Out[53]:
```
[Timestamp('2019-01-01 00:00:00')]
```

In [54]:
```
get_dates('every week until July 10th')[:10]
```

Out[54]:
```
[Timestamp('2018-12-03 00:00:00'),
 Timestamp('2018-12-10 00:00:00'),
 Timestamp('2018-12-17 00:00:00'),
 Timestamp('2018-12-24 00:00:00'),
 Timestamp('2018-12-31 00:00:00'),
 Timestamp('2019-01-07 00:00:00'),
 Timestamp('2019-01-14 00:00:00'),
 Timestamp('2019-01-21 00:00:00'),
 Timestamp('2019-01-28 00:00:00'),
 Timestamp('2019-02-04 00:00:00')]
```

In [55]:
```
# get_dates("this won't work")
```


i love bubble tea

In [56]:
```
dates = get_dates('every weekday')
dates[:10]  # first ten instances of the recurrance rule
```

Out[56]:
```
[Timestamp('2018-12-03 00:00:00'),
 Timestamp('2018-12-04 00:00:00'),
 Timestamp('2018-12-05 00:00:00'),
 Timestamp('2018-12-06 00:00:00'),
 Timestamp('2018-12-07 00:00:00'),
 Timestamp('2018-12-10 00:00:00'),
 Timestamp('2018-12-11 00:00:00'),
 Timestamp('2018-12-12 00:00:00'),
 Timestamp('2018-12-13 00:00:00'),
 Timestamp('2018-12-14 00:00:00')]
```

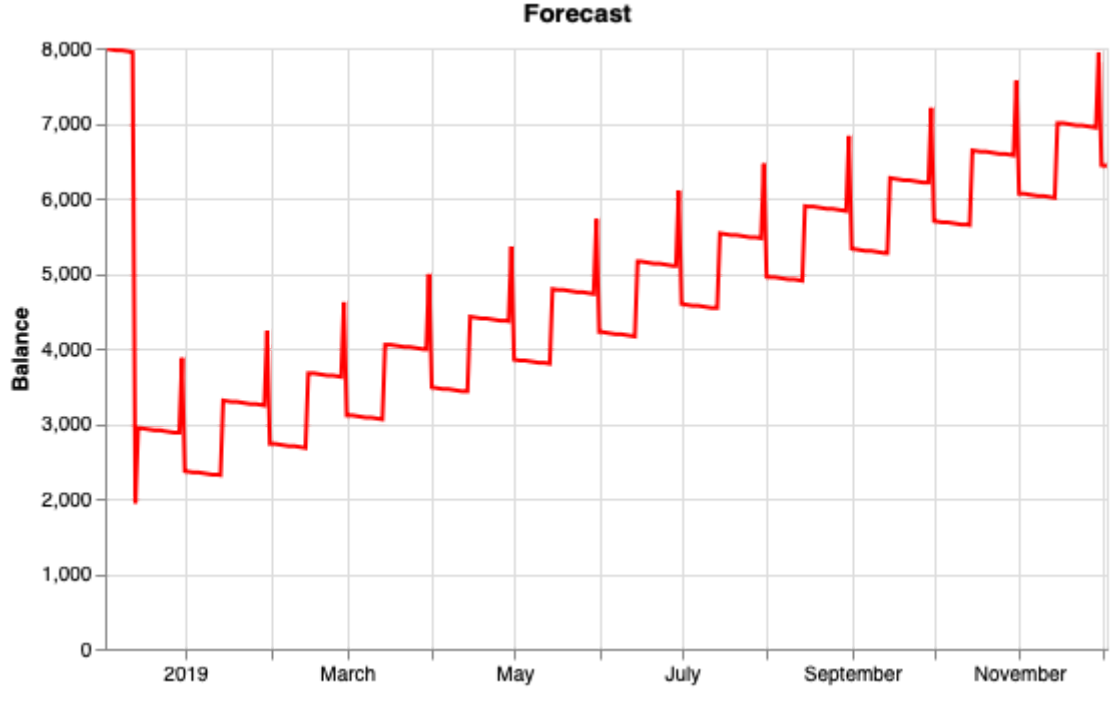
In [57]:
```
pd.DatetimeIndex(pd.Series(dates))[:10]
```

Out[57]:
```
DatetimeIndex(['2018-12-03', '2018-12-04', '2018-12-05', '2018-12-06',
               '2018-12-07', '2018-12-10', '2018-12-11', '2018-12-12',
               '2018-12-13', '2018-12-14'],
              dtype='datetime64[ns]', freq=None)
```

In [58]:
```
dates = get_dates('every weekday')

bbt = pd.DataFrame(
    data={'bubble_tea': -6},
    index=pd.DatetimeIndex(pd.Series(dates))
)
```

In [59]:
```
calendar = pd.concat([calendar, bbt], axis=1).fillna(0)
```

In [60]:
```
calendar = update_totals(calendar)
plot_budget(calendar)
```



YAML

```
!pip install pyyaml
```

In [61]:

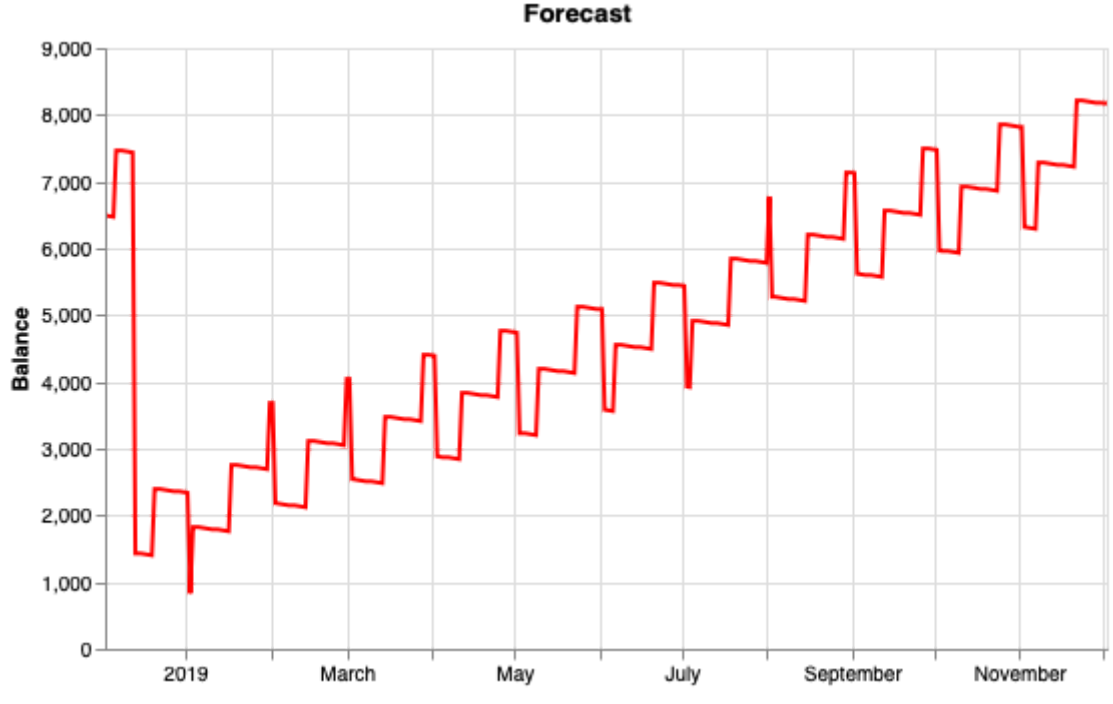
```
import yaml

budget = yaml.load('''
bank:
    frequency: today
    amount: 8000
income:
    frequency: every 2 weeks on Friday
    amount: 1000
rent:
    frequency: every month
    amount: -1500
mauritius:
    frequency: 2018-12-14
    amount: -6000
bubble_tea:
    frequency: every weekday
    amount: -7
''')
```

In [62]:
```
budget
```

Out[62]:
```
{'bank': {'frequency': 'today', 'amount': 8000},
 'income': {'frequency': 'every 2 weeks on Friday', 'amount': 1000},
 'rent': {'frequency': 'every month', 'amount': -1500},
 'mauritius': {'frequency': datetime.date(2018, 12, 14), 'amount': -6000},
 'bubble_tea': {'frequency': 'every weekday', 'amount': -7}}
```

In [63]:
```
calendar = pd.DataFrame(index=pd.date_range(start, end))

for k, v in budget.items():
    frequency = v.get('frequency')
    amount = v.get('amount')
    dates = get_dates(frequency)
    i = pd.DataFrame(
        data={k: amount},
        index=pd.DatetimeIndex(pd.Series(dates))
    )
    calendar = pd.concat([calendar, i], axis=1).fillna(0)

calendar['total'] = calendar.sum(axis=1)
calendar['cum_total'] = calendar['total'].cumsum()
```

In [64]:
```
plot_budget(calendar)
```



In [65]:

```
def build_calendar(budget):

    calendar = pd.DataFrame(index=pd.date_range(start, end))

    for k, v in budget.items():
        frequency = v.get('frequency')
        amount = v.get('amount')
        dates = get_dates(frequency)
        i = pd.DataFrame(
            data={k: amount},
            index=pd.DatetimeIndex(pd.Series(dates))
        )
        calendar = pd.concat([calendar, i], axis=1).fillna(0)

    calendar['total'] = calendar.sum(axis=1)
    calendar['cum_total'] = calendar['total'].cumsum()

    return calendar
```

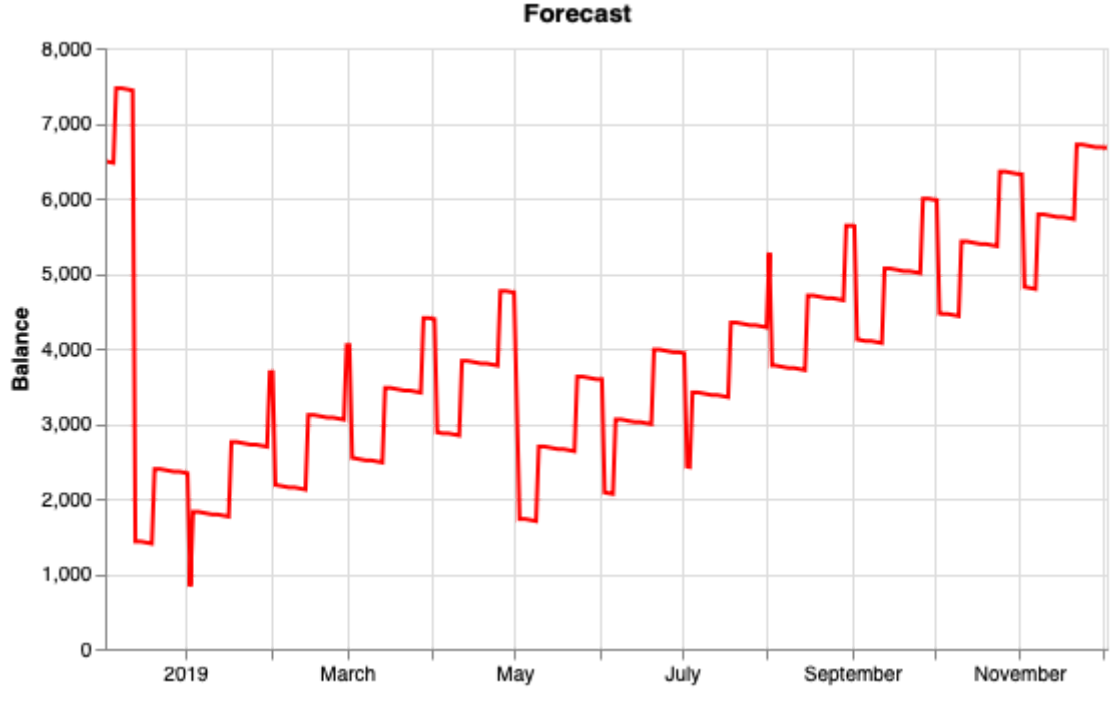

53

In [66]:

```
budget = yaml.load('''
bank:
    frequency: today
    amount: 8000
income:
    frequency: every 2 weeks on Friday
    amount: 1000
rent:
    frequency: every month
    amount: -1500
mauritius:
    frequency: 2018-12-14
    amount: -6000
bubble_tea:
    frequency: every weekday
    amount: -7
england:
    frequency: 2019-05-02
    amount: -1500
''')
```

In [67]:
```
calendar = build_calendar(budget)
```

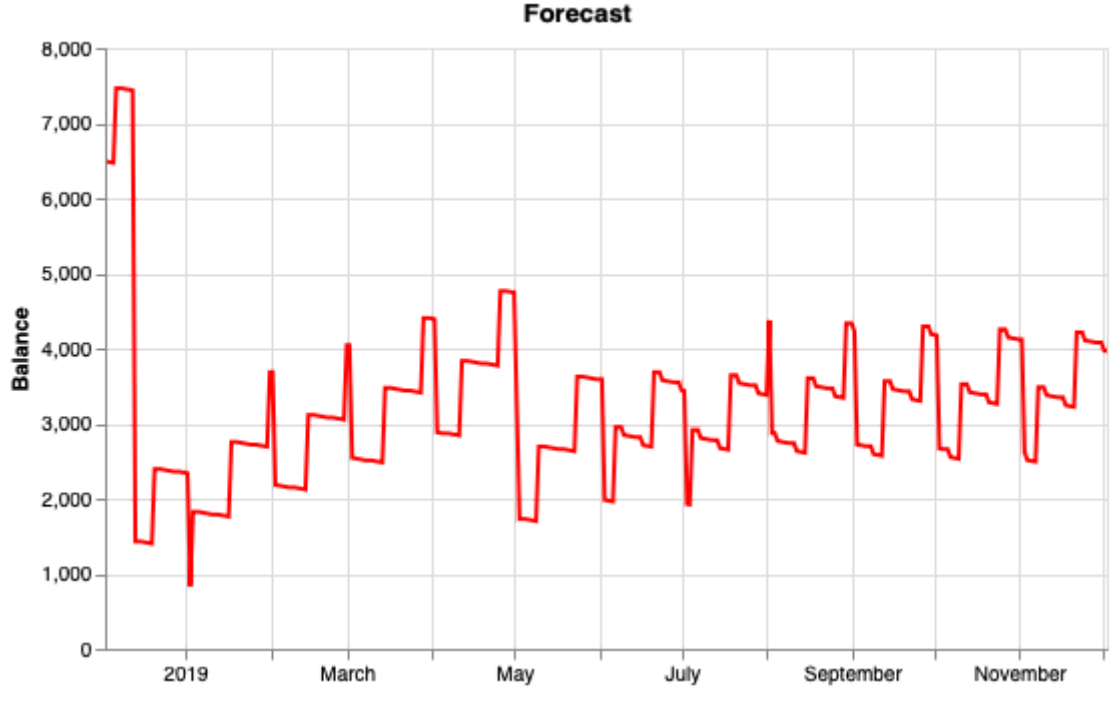
In [68]:
```
plot_budget(calendar)
```



In [69]:

```
budget = yaml.load('''
bank:
    frequency: today
    amount: 8000
income:
    frequency: every 2 weeks on Friday
    amount: 1000
rent:
    frequency: every month
    amount: -1500
mauritius:
    frequency: 2018-12-14
    amount: -6000
bubble_tea:
    frequency: every weekday
    amount: -7
england:
    frequency: 2019-05-02
    amount: -1500
savings:
    frequency: every Monday starting in June
    amount: -100
''')
```

In [70]:
```
calendar = build_calendar(budget)
plot_budget(calendar)
```



🔥 demo


Thank
you!

- Twitter: `@maxhumber`
- LinkedIn: `/in/maxhumber`