A 30-year fixed rate at 4%. The loan will amortize over 30 years.

Monthly Payment = $\frac{LoanAmount * \frac{InterestRate}{12}}{1-(1+\frac{InterestRate}{12})^{(-LoanTermInMonths)}}$

So when filled with the according values it comes like this:

Monthly Payment = $\frac{1{,}000{,}000 * \frac{0.04}{12}}{1-(1+\frac{0.04}{12})^{(-30*12)}}$

Monthly Payment = $4,774.15

Here goes the code implementation for a 30-year fixed rate at 4%

In [ ]:
```
# libraries

import pandas as pd
import numpy as np
```

In [ ]:

```
# Loan details
loan_amount = 1000000
interest_rate = 0.04
loan_term = 30

# Calculate the fixed monthly payment
monthly_interest_rate = interest_rate / 12
num_payments = loan_term * 12
fixed_payment = loan_amount * (monthly_interest_rate * np.power(1 + mont


# Initialize the amortization schedule dataframe
# amortization_schedule = pd.DataFrame(columns=['Month', 'Fixed Payment
amortization_schedule = []
principal_balance = loan_amount

# Populate the amortization schedule
for month in range(0, num_payments + 1):
    if month == 0:
        amortization_schedule.append({
            'Month': month,
            'Fixed Payment': 0,
            'Principal Paydown': 0,
            'Interest Applied': 0,
            'Principal Balance': principal_balance})
    else:
        interest_applied = principal_balance * monthly_interest_rate
        principal_paydown = fixed_payment - interest_applied
        principal_balance -= principal_paydown
        amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})

# Calculate the total interest paid
df = pd.DataFrame(amortization_schedule)
total_interest_paid = df['Interest Applied'].sum()

print("Total Interest Paid: $", round(total_interest_paid, 2))
df
```

```
Total Interest Paid: $ 718695.06
```

Out[2]:

	Month	Fixed Payment	Principal Paydown	Interest Applied	Principal Balance
0	0	0.000000	0.000000	0.000000	1.000000e+06
1	1	4774.152955	1440.819621	3333.333333	9.985592e+05
2	2	4774.152955	1445.622353	3328.530601	9.971136e+05
3	3	4774.152955	1450.441095	3323.711860	9.956631e+05
4	4	4774.152955	1455.275898	3318.877056	9.942078e+05
...	...	...	...	...	...
356	356	4774.152955	4695.373283	78.779671	1.893853e+04
357	357	4774.152955	4711.024527	63.128427	1.422750e+04
358	358	4774.152955	4726.727943	47.425012	9.500776e+03
359	359	4774.152955	4742.483702	31.669252	4.758292e+03
360	360	4774.152955	4758.291981	15.860973	3.894911e-08

361 rows × 5 columns

Ideal customer:

A borrower with a stable income, who plans to stay in the home long-term and prefers predictable monthly payments. This type of customer typically has a lower risk tolerance and appreciates the certainty of a fixed interest rate over the entire loan term. They might have a moderate income level, making the lower monthly payments of a 30-year mortgage more manageable compared to shorter-term loans with higher monthly payments

The code provided in a previous homework assignment for a 30-year fixed rate at 4% already uses a Python list, which is considered to be the best practice. Here is a detailed analysis:

Creating a data frame for the cash flow sheet requires O(n) time complexity and O(n) space complexity, where n is the number of rows required by the homework. This is because at least one full iteration of the loop is necessary to create the data frame using a code line like pd.DataFrame(amortization_schedule) or any other type of presentation of the sheet.

The previous solution only used O(1*n) time and space complexity, which is considered to be following the best practice. However, some might suggest using list comprehension to improve the speed under the same complexity due to the nature of Python.

In my opinion, using list comprehension would not be ideal in cases where the operation involves multiple nested branching conditions. Regardless of the potential difficulty in reading the code with list comprehension, the loop in this case contains a conditional statement and multiple branching conditions

A 20-year fixed rate at 2.5%. The loan will amortize over 20 years.

Monthly Payment = $\frac{LoanAmount * \frac{InterestRate}{12}}{1-(1+\frac{InterestRate}{12})^{(-LoanTermInMonths)}}$

So when filled with the according values it comes like this:

Monthly Payment = $\frac{1{,}000{,}000 * \frac{0.025}{12}}{1-(1+\frac{0.025}{12})^{(-20*12)}}$

Monthly Payment = $ 5299.03

Here goes the code implementation for a 20-year fixed rate at 2.5%

In [ ]:

```
# Loan details
loan_amount = 1000000
interest_rate = 0.025
loan_term = 20

# Calculate the fixed monthly payment
monthly_interest_rate = interest_rate / 12
num_payments = loan_term * 12
fixed_payment = loan_amount * (monthly_interest_rate * np.power(1 + mont


# Initialize the amortization schedule dataframe
# amortization_schedule = pd.DataFrame(columns=['Month', 'Fixed Payment'
amortization_schedule = []
principal_balance = loan_amount

# Populate the amortization schedule
for month in range(0, num_payments + 1):
    if month == 0:
        amortization_schedule.append({
            'Month': month,
            'Fixed Payment': 0,
            'Principal Paydown': 0,
            'Interest Applied': 0,
            'Principal Balance': principal_balance})
    else:
        interest_applied = principal_balance * monthly_interest_rate
        principal_paydown = fixed_payment - interest_applied
        principal_balance -= principal_paydown
        amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})

# Calculate the total interest paid
df = pd.DataFrame(amortization_schedule)
total_interest_paid = df['Interest Applied'].sum()

print("Total Interest Paid: $", round(total_interest_paid, 2))
df
```

```
Total Interest Paid: $ 271766.94
```

Out[3]:

	Month	Fixed Payment	Principal Paydown	Interest Applied	Principal Balance
0	0	0.00000	0.000000	0.000000	1.000000e+06
1	1	5299.02893	3215.695597	2083.333333	9.967843e+05
2	2	5299.02893	3222.394963	2076.633968	9.935619e+05
3	3	5299.02893	3229.108286	2069.920645	9.903328e+05
4	4	5299.02893	3235.835595	2063.193336	9.870970e+05
...	...	...	...	...	...
236	236	5299.02893	5244.174031	54.854899	2.108618e+04
237	237	5299.02893	5255.099394	43.929537	1.583108e+04
238	238	5299.02893	5266.047517	32.981413	1.056503e+04
239	239	5299.02893	5277.018450	22.010481	5.288012e+03
240	240	5299.02893	5288.012238	11.016692	6.353002e-08

241 rows × 5 columns

This is a fixed-rate mortgage and is for 20 years which makes it suitable for people with serious income like the people working in the Silicon Valley and especially the IT people. It is suitable for them as an investment home because they have steady and enormous (compared to the rest of the society) income cash flows. Thus, this is a suitable investment for them because they can relatively quickly pay the mortgage and furthermore if this is their second home they can even put it as a rental home and manage to repay it even quicker with the additional income from the rent.

A 7-1 Adjustable Rate Mortgage (ARM) that varies according to rates.

Type *Markdown* and LaTeX: α^2

Monthly Payment = $\frac{LoanAmount * \frac{InterestRate}{12}}{1-(1+\frac{InterestRate}{12})^{(-LoanTermInMonths)}}$

So when filled with the according values it comes like this:

Monthly Payment = $\frac{1,000,000 * \frac{0.0754}{12}}{1-(1+\frac{0.0754}{12})^{(-30*12)}}$

Monthly Payment = \$ 7019.56

Here goes the code implementation for a 7-1 ARM 30-year mortgage with initial rate at 7.54%. Changing interest rate will change monthly payments. 30 year interests rates are given in the code below. Monthly payments were taken from excel GWP1.

In [ ]:

```
#Loan details
loan_amount=1000000
interest_rates= {
    "Year": [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,
    "Rates": [7.54,7.54,7.54,7.54,7.54,7.54,7.54,9.64,11.20,13.74,16.63,
}
loan_term=30
# Calculate the monthly payment for initial 7 years
monthly_interest_rate = float(interest_rates["Rates"][0]) / (12*100)
num_payments = loan_term * 12
fixed_payment = loan_amount * (monthly_interest_rate * np.power(1 + mont


# Initialize the amortization schedule dataframe
# amortization_schedule = pd.DataFrame(columns=['Month', 'Fixed Payment'
amortization_schedule = []
principal_balance = loan_amount
# Populate the amortization schedule for initial 7 year
for month in range(0, 7*12 + 1):
    if month == 0:
        amortization_schedule.append({
            'Month': month,
            'Fixed Payment': 0,
            'Principal Paydown': 0,
            'Interest Applied': 0,
            'Principal Balance': principal_balance})
    else:
        interest_applied = principal_balance * monthly_interest_rate
        principal_paydown = fixed_payment - interest_applied
        principal_balance = principal_balance - principal_paydown
        amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})
#monthly payment in year between 7 and 8.
monthly_interest_rate=float(interest_rates["Rates"][7]) / (12*100)
num_payments=num_payments-84
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(7*12+1,8*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})
#monthly payment in year between 8 and 9.
monthly_interest_rate=float(interest_rates["Rates"][8]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(8*12+1,9*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
```

```
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})
#monthly payment in year between 9 and 10.
monthly_interest_rate=float(interest_rates["Rates"][9]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(9*12+1,10*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})
#monthly payment in year between 10 and 11.
monthly_interest_rate=float(interest_rates["Rates"][10]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(10*12+1,11*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})
#monthly payment in year between 11 and 12.
monthly_interest_rate=float(interest_rates["Rates"][11]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(11*12+1,12*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})
#monthly payment in year between 12 and 13.
monthly_interest_rate=float(interest_rates["Rates"][12]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(12*12+1,13*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
```

```
    amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})
#monthly payment in year between 13 and 14.
monthly_interest_rate=float(interest_rates["Rates"][13]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(13*12+1,14*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})
#monthly payment in year between 14 and 15.
monthly_interest_rate=float(interest_rates["Rates"][14]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(14*12+1,15*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})
#monthly payment in year between 15 and 16.
monthly_interest_rate=float(interest_rates["Rates"][15]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(15*12+1,16*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})
#monthly payment in year between 16 and 17.
monthly_interest_rate=float(interest_rates["Rates"][16]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(16*12+1,17*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
```

```
                'Month': month,
                'Fixed Payment': fixed_payment,
                'Principal Paydown': principal_paydown,
                'Interest Applied': interest_applied,
                'Principal Balance': principal_balance})
#monthly payment in year between 17 and 18.
monthly_interest_rate=float(interest_rates["Rates"][17]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(17*12+1,18*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
                'Month': month,
                'Fixed Payment': fixed_payment,
                'Principal Paydown': principal_paydown,
                'Interest Applied': interest_applied,
                'Principal Balance': principal_balance})
#monthly payment in year between 18 and 19.
monthly_interest_rate=float(interest_rates["Rates"][18]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(18*12+1,19*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
                'Month': month,
                'Fixed Payment': fixed_payment,
                'Principal Paydown': principal_paydown,
                'Interest Applied': interest_applied,
                'Principal Balance': principal_balance})
#monthly payment in year between 19 and 20.
monthly_interest_rate=float(interest_rates["Rates"][19]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(19*12+1,20*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
                'Month': month,
                'Fixed Payment': fixed_payment,
                'Principal Paydown': principal_paydown,
                'Interest Applied': interest_applied,
                'Principal Balance': principal_balance})

#monthly payment in year between 20 and 21.
monthly_interest_rate=float(interest_rates["Rates"][20]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(20*12+1,21*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
```

```
                'Month': month,
                'Fixed Payment': fixed_payment,
                'Principal Paydown': principal_paydown,
                'Interest Applied': interest_applied,
                'Principal Balance': principal_balance})

#monthly payment in year between 21 and 22.
monthly_interest_rate=float(interest_rates["Rates"][21]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(21*12+1,22*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
                'Month': month,
                'Fixed Payment': fixed_payment,
                'Principal Paydown': principal_paydown,
                'Interest Applied': interest_applied,
                'Principal Balance': principal_balance})

#monthly payment in year between 22 and 23.
monthly_interest_rate=float(interest_rates["Rates"][22]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(22*12+1,23*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
                'Month': month,
                'Fixed Payment': fixed_payment,
                'Principal Paydown': principal_paydown,
                'Interest Applied': interest_applied,
                'Principal Balance': principal_balance})

#monthly payment in year between 23 and 24.
monthly_interest_rate=float(interest_rates["Rates"][23]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(23*12+1,24*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
                'Month': month,
                'Fixed Payment': fixed_payment,
                'Principal Paydown': principal_paydown,
                'Interest Applied': interest_applied,
                'Principal Balance': principal_balance})

 #monthly payment in year between 24 and 25.
monthly_interest_rate=float(interest_rates["Rates"][24]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(24*12+1,25*12+1):
    interest_applied = principal_balance * monthly_interest_rate
```

```
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})

#monthly payment in year between 25 and 26.
monthly_interest_rate=float(interest_rates["Rates"][25]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(24*12+1,25*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})

#monthly payment in year between 26 and 27.
monthly_interest_rate=float(interest_rates["Rates"][26]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(25*12+1,26*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})

#monthly payment in year between 27 and 28.
monthly_interest_rate=float(interest_rates["Rates"][27]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(26*12+1,27*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})

#monthly payment in year between 28 and 29.
monthly_interest_rate=float(interest_rates["Rates"][28]) / (12*100)
num_payments=num_payments-12
```

```
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(27*12+1,28*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})

#monthly payment in year between 29 and 30.
monthly_interest_rate=float(interest_rates["Rates"][29]) / (12*100)
num_payments=num_payments-12
fixed_payment = principal_balance * (monthly_interest_rate * np.power(1
for month in range(28*12+1,29*12+1):
    interest_applied = principal_balance * monthly_interest_rate
    principal_paydown = fixed_payment - interest_applied
    principal_balance = principal_balance - principal_paydown
    amortization_schedule.append({
            'Month': month,
            'Fixed Payment': fixed_payment,
            'Principal Paydown': principal_paydown,
            'Interest Applied': interest_applied,
            'Principal Balance': principal_balance})

# Calculate the total interest paid
df = pd.DataFrame(amortization_schedule)
total_interest_paid = df['Interest Applied'].sum()

print("Total Interest Paid: $", round(total_interest_paid, 2))

df
```

```
Total Interest Paid: $ 2151793.54
```

Out[4]:

	Month	Fixed Payment	Principal Paydown	Interest Applied	Principal Balance
0	0	0.000000	0.000000	0.000000	1.000000e+06
1	1	7019.555555	736.222222	6283.333333	9.992638e+05
2	2	7019.555555	740.848152	6278.707404	9.985229e+05
3	3	7019.555555	745.503148	6274.052408	9.977774e+05
4	4	7019.555555	750.187392	6269.368163	9.970272e+05
...	...	...	...	...	...
356	344	8186.973821	7917.810137	269.163683	3.220597e+04
357	345	8186.973821	7970.925447	216.048374	2.423504e+04
358	346	8186.973821	8024.397072	162.576749	1.621065e+04
359	347	8186.973821	8078.227402	108.746419	8.132419e+03
360	348	8186.973821	8132.418844	54.554976	-3.328751e-10

361 rows × 5 columns

An ideal customer for a 30-year, 7-1 adjustable rate mortgage is a person who thinks that current interest rate is high and expects interest rate to fall in the future. This mortgage has an interest rate risk when it rises. Therefore, this mortgage is not for everyone. You should have enough savings, so you can continue paying mortgage payments even in a high interest rate environment. You should also have steady income, so you can make monthly payments.

REFERENCES:

1. “What Is a Mortgage? Types, How They Work, and Examples.” Investopedia, https://www.investopedia.com/terms/m/mortgage.asp (https://www.investopedia.com/terms/m/mortgage.asp). Accessed 4 April 2023.
2. “Mortgage Comparison & Mortgage Rates.” MoneySuperMarket, https://www.moneysupermarket.com/mortgages/ (https://www.moneysupermarket.com/mortgages/). Accessed 4 April 2023.
3. Mortgage Calculator, https://www.mortgagecalculator.org/ (https://www.mortgagecalculator.org/). Accessed 4 April 2023
4. “Mortgage Calculator.” Bankrate, https://www.bankrate.com/mortgages/mortgage- (https://www.bankrate.com/mortgages/mortgage-) calculator/. Accessed 4 April 2023.
5. De Vita, Suzanne. “What Is A 7/1 Adjustable-Rate Mortgage?” Bankrate, 18 May 2022, https://www.bankrate.com/mortgages/what-is-a-7-1-arm/ (https://www.bankrate.com/mortgages/what-is-a-7-1-arm/). Accessed 4 April
6. “Compare Today's 7/1 ARM Mortgage Rates.” SmartAsset.com, https://smartasset.com/mortgage/7-1-arm-mortgage-rates (https://smartasset.com/mortgage/7-1-arm-mortgage-rates). Accessed 4 April 2023.
7. “Fixed-Rate vs. Adjustable-Rate Mortgages: What's the Difference?” Investopedia, https://www.investopedia.com/mortgage/mortgage-rates/fixed-versus-adjustable- (https://www.investopedia.com/mortgage/mortgage-rates/fixed-versus-adjustable-) rate/. Accessed 4 April 2023.

8. Lewis, Holden. “Compare Today’s 7/1 ARM Mortgage Rates.” NerdWallet, https://www.nerdwallet.com/mortgages/mortgage-rates/7-1-arm (https://www.nerdwallet.com/mortgages/mortgage-rates/7-1-arm). Accessed 4 April
9. NumPy quickstart — NumPy v1.25.dev0 Manual.
10. Introduction to NumPy (w3schools.com).
11. NumPy Illustrated: The Visual Guide to NumPy | by Lev Maximov | Better Programming.
12. https://res.cloudinary.com/dyd911kmh/image/upload/v1676302459/Marketing/Blog/Numpy_((https://res.cloudinary.com/dyd911kmh/image/upload/v1676302459/Marketing/Blog/Numpy_
13. https://intellipaat.com/blog/tutorial/python-tutorial/numpy-cheat-sheet/ (https://intellipaat.com/blog/tutorial/python-tutorial/numpy-cheat-sheet/)
14. https://web.itu.edu.tr/iguzel/files/Python_Cheat_Sheets.pdf (https://web.itu.edu.tr/iguzel/files/Python_Cheat_Sheets.pdf)
15. https://docs.python.org/3/tutorial/datastructures.html (https://docs.python.org/3/tutorial/datastructures.html)
16. https://www.w3schools.com/python/python_dictionaries.asp (https://www.w3schools.com/python/python_dictionaries.asp)
17. https://www.w3schools.com/python/python_dictionaries_access.asp (https://www.w3schools.com/python/python_dictionaries_access.asp)
18. https://www.w3schools.com/python/python_dictionaries_change.asp (https://www.w3schools.com/python/python_dictionaries_change.asp)
19. https://www.w3schools.com/python/python_dictionaries_add.asp (https://www.w3schools.com/python/python_dictionaries_add.asp)
20. https://www.w3schools.com/python/python_dictionaries_remove.asp (https://www.w3schools.com/python/python_dictionaries_remove.asp)
21. https://www.w3schools.com/python/python_dictionaries_loop.asp

In [ ]: