

Work on project. Stage 4/4: Differentiate payment

290 users solved this problem. Latest completion was 11 minutes ago.

Project: [Credit Calculator](#)

Hard 1 hour ?

Description

Finally, let's add to our calculator the capacity to compute the differentiated payment. In such a kind of payment where the part for reducing the credit principal is constant. Another part of the payment is for interest repayment and it reduces during the credit term. It means that the payment is different each month. Let's look at the formula:

$$D_m = \frac{P}{n} + i * \left(P - \frac{P * (m - 1)}{n} \right)$$

Where:

D_m = mth differentiated payment;

P = the credit principal;

i = nominal interest rate. Usually, it's 1/12 of the annual interest rate. And usually, it's a floating value, not a percentage. For example, if we our annual interest rate = 12%, then $i = 0.01$.

n = Number of payments. Usually, it's the count of months.

m = current period.

As you can see, the user has to input a lot of parameters. So it might be convenient to use command-line arguments.

Suppose you used to run your credit calculator via command line like this:

```
1 | python credit_calc.py
```

Using command-line arguments you can run your program this way:

```
1 | python credit_calc.py --type=diff --principal=1000000 --periods=10 --interest=10
```

In that case, your program can get different values and not ask the user to input them. It can be useful when you are developing your program and trying to find a mistake and want to run the program with the same parameters again and again. Also, it's convenient if you made a mistake in a single parameter. You don't have to input all other values again.

To confidently work with command-line arguments in Python, check out a [tutorial](#).

Objectives

At this stage, it is required to implement these features:

- the calculation of differentiated payment. To do this, the user may run the program specifying interest, count of periods and credit principal.
- a capacity to calculate the same values as in the previous stage for annuity payment (principal, count of periods and value of the payment). A user specifies all known parameters with command-line arguments, while a single parameter will be unknown. This is the value the user wants to calculate.
- handling of invalid parameters. It's a good idea to show an error message `Incorrect parameters` in case of invalid parameters (they are discussed in detail below).

The final version of your program is supposed to work from the command line and parse the following parameters:

10 / 10 Prerequisites

- ✓ [Command line overview](#) Stage 4
- ✓ [Type casting](#) 15 ★ ... Stage 4
- ✓ [List](#) 25 ★ ... Stage 4
- ✓ [Indexes](#) 18 ★ ... Stage 4
- ✓ [Declaring a function](#) 18 ★ ... Stage 4

Show all

- `--type`, which indicates the type of payments: `"annuity"` or `"diff"` (differentiated). If `--type` is specified neither as `"annuity"` nor as `"diff"`, or it is not specified at all, show the error message.

```
1 > python credit_calc.py --principal=1000000 --periods=60 --interest=10
2 Incorrect parameters
```

- `--payment`, that is a monthly payment. For `--type=diff` the payment is different each month, so we can't calculate periods or principal, therefore, its combination with `--payment` is invalid, too:

```
1 > python credit_calc.py --type=diff --principal=1000000 --interest=10 --payment=100000
2 Incorrect parameters
```

- `--principal` is used for calculations of both types of payment. You can get its value knowing the interest, annuity payment and periods.
- `--periods` parameter denotes the number of months and/or years needed to repay the credit. It's calculated based on the interest, annuity payment and principal.
- `--interest` is specified without a percent sign. Note that it may accept a floating-point value. Our credit calculator can't calculate the interest, so these parameters are incorrect:

```
1 > python credit_calc.py --type=annuity --principal=100000 --payment=10400 --periods=8
2 Incorrect parameters
```

Let's make a comment. You might have noticed that for differentiated payments you will need 4 out of 5 parameters (excluding payment), and the same is true for annuity payments (missing either periods, payment or principal). Thus, when less than four parameters are given, you should display the error message too:

```
1 > python credit_calc.py --type=annuity --principal=1000000 --payment=104000
2 Incorrect parameters
```

Another case when you should output this message is negative values. We can't work with these!

```
1 > python credit_calc.py --type=diff --principal=30000 --periods=-14 --interest=10
2 Incorrect parameters
```

Finally, don't forget to compute the value of overpayment, and you'll have your real credit calculator!

Examples

Let's look at the first example. The greater-than symbol followed by space (`>`) represents the user input.

Example 1: *calculate differentiated payments*

```
1 > python credit_calc.py --type=diff --principal=1000000 --periods=10 --interest=10
2 Month 1: paid out 108334
3 Month 2: paid out 107500
4 Month 3: paid out 106667
5 Month 4: paid out 105834
6 Month 5: paid out 105000
7 Month 6: paid out 104167
8 Month 7: paid out 103334
9 Month 8: paid out 102500
1
0 Month 9: paid out 101667
1
1 Month 10: paid out 100834
1
2
1
3 Overpayment = 45837
```

In this example, the user wants to take a credit with differentiated payments. You know the principal, the count of periods and interest, which are 1,000,000, 10 months and 10% respectively.

The calculator should calculate payments for all 10 months. Let's look at the formula above. In this case:

$$P = 100000$$

$$n = 10$$

$$i = \frac{\text{interest}}{12 * 100\%} = \frac{10\%}{12 * 100\%} = 0.008333...$$

Then, let's find the payment for the first month ($m = 1$):

$$D_1 = \frac{P}{n} + i * \left(P - \frac{P * (m - 1)}{n} \right) = \frac{1000000}{10} + 0.008333... * \left(1000000 - \frac{1000000 * (1 - 1)}{10} \right) = 108333.333...$$

The second month ($m = 2$):

$$D_1 = \frac{P}{n} + i * \left(P - \frac{P * (m - 1)}{n} \right) = \frac{1000000}{10} + 0.008333... * \left(1000000 - \frac{1000000 * (2 - 1)}{10} \right) = 107500$$

The third month ($m = 3$):

$$D_1 = \frac{P}{n} + i * \left(P - \frac{P * (m - 1)}{n} \right) = \frac{1000000}{10} + 0.008333... * \left(1000000 - \frac{1000000 * (3 - 1)}{10} \right) = 106666.666...$$

And so on. You can see other monthly payments above.

Your credit calculator should output monthly payments **for every month** like at the first stage. Also, round up all floating-point values.

Finally, your credit calculator should add up all the payments and subtract the credit principal so that you'll get the overpayment.

Example 2: *find an annuity payment for the 60-month (or 5-year) credit with the principal 1,000,000 and 10% interest*

```
1 > python credit_calc.py --type=annuity --principal=1000000 --periods=60 --interest=10
2 Your annuity payment = 21248!
3 Overpayment = 274880
```

Example 3: *less than four arguments are given*

```
1 > python credit_calc.py --type=diff --principal=1000000 --payment=104000
2 Incorrect parameters.
```

Example 4: *calculate differentiated payments given the principal 500,000, the period of 8 months and the interest rate 7.8%*

```
1 > python credit_calc.py --type=diff --principal=500000 --periods=8 --interest=7.8
2 Month 1: paid out 65750
3 Month 2: paid out 65344
4 Month 3: paid out 64938
5 Month 4: paid out 64532
6 Month 5: paid out 64125
7 Month 6: paid out 63719
8 Month 7: paid out 63313
9 Month 8: paid out 62907
1
0
1
1 Overpayment = 14628
```

Example 5: *calculate the principal for a user paying 8,722 per month for 120 months (10 years) with the interest 5.6%*


```
1 > python credit_calc.py --type=annuity --payment=8722 --periods=120 --interest=5.6
2 Your credit principal = 800018!
3 Overpayment = 246622
```

Example 6: *figure out how much time a user needs to repay the credit with 500,000 principal, 23,000 monthly payment and 7.8% interest*

```
1 > python credit_calc.py --type=annuity --principal=500000 --payment=23000 --interest=7.8
2 You need 2 years to repay this credit!
3 Overpayment = 52000
```

[Code Editor](#) [IDE](#)

• IDE / Checking the plugin's status


 [Solve in IDE](#)

✓ **Correct**

22 users liked this problem. **5** didn't like it. **What about you?**

[Continue](#)

 [View solutions \(49\)](#)

 [Show discussion \(100\)](#)