I'm not a bot

Rounding a number is a common operation we often need to carry to ensure that the value in slightly changed to ensure its readability. In this tutorial, we'll explain how you can round a number in Python and explain some common use cases and how we can solve them. Python comes with a built-in function, round() that takes two parameters: a number and the number of digits we need after the decimal point. If no number of digits is specified, the function will round up and down to the closest integer. print(round(3.2)) print(round(3.269,2))Code language: Python (python) Output: Code language: Python (python) The floating-point precision, represented by the second parameter in our function, will often be left empty or with a small number; otherwise you'll end up removing the main benefits of this function. One surprising behavior from this function is that if we run round(3.675, 2), it will return 3.67 instead of the expected 3.68. This is not a bug and the reason is explained in the official documentation, but it is counterintuitive, and we must be aware of it. While this function comes handy, we can't control whether we round up or down a number. In some cases, you'll want to round down, even if the number is 3.7 for instance. To get this behavior, we should use the math library, coming with two handy methods: math.floor() will return the largest integer less than or equal to xmath.ceil() will return the smallest integer greater than or equal to x Both takes only one argument, and you can't get a decimal value when you round up or down using this library. import math print(math.floor(3.7)) print(math.ceil(3.7))Code language: Python (python) Output: Code language: Python (python) If we need to round to the nearest multiple of 10, 100 or even 1000 (most common cases), we can use the round() function or the math library with some tweaks. In our example at the beginning of this article, we explained the second parameters of the round() function is used to indicate the number of digits we want to keep. If you use a negative value, you'd actually let the function know that: -1: you want to round to the closest 10 -2: you wish to round to the closest 100-3: you wish to round to the closest 1000 etc... print(round(987,-1)) print(round(1010,-2)) print(round(1987,-3))Code language: Python (python) Output: Code language: Python (python) Awesome, right? Now let's see how we can achieve the same using the math library. The logic is less straightforward (in my opinion) but can be achieved with a reduced number of lines. Let's do it with 990. Two steps: We first calculate the math.ceil() or math.floor(), divided by the multiple we want to get. Let's assume I want a multiple of 100 here: print(math.ceil(990/100)) Code language: PHP (php) We then multiply it by the multiple again to get the closest multiple of 100 print(math.ceil(990/100)*100) Code language: PHP (php) The same logic can be applied with math.floor() obviously: it will depend on whether you want to round up or down. Sometimes, we wish to print a number without a decimal. Even if we don't want to modify the value of the variable holding this number. To truncate a float in Python, you just need to use the format() method. value = 132.8747474 print('the value is equal to {}'.format(round(value, 2)))Code language: Python (python) Output: the value is equal to 132.87Code language: Python (python) Rounding a number simplifies it while keeping its value as close as possible to the original. Python provides various methods to round numbers, depending on how we want to handle the precision or rounding behavior. In this article, we'll cover the most commonly used techniques for rounding numbers in Python, along with examples. For example, Input is 3.5 then Output should be 4.Using Built-in round() FunctionPython's built-in round() function rounds a number to a given precision. If no precision is specified, it rounds to the nearest integer. python print(round(11)) print(round(22.7)) print(round(4.465, 2)) print(round(4.476, 2)) print(round(4.473, 2)) Output11 23 4.46 4.48 4.47 Explanation:round(x) rounds x to the nearest integer.round(x, n) rounds x to n decimal places.Using Truncation (Cutting Off Digits)Truncation cuts off digits after a specified number of decimal places without rounding. It works for both positive and negative numbers. python def truncate(num, dec=0): mult = 10 ** dec return int(num * mult) / mult print(truncate(16.5)) print(truncate(-3.853, 1)) print(truncate(3.815, 2)) print(truncate(346.8, -1)) print(truncate(-2947.48, -3)) Output16.0 -3.8 3.81 340.0 -2000.0 Explanation:Multiplies by 10^dec, truncates, and divides back.Useful when you only want to chop off digits, not round.Using math.ceil() and math.floor()Python's math module provides two essential functions:ceil(x): Rounds up to the nearest integer.floor(x): Rounds down to the nearest integer. python import math print(math.ceil(4.2)) print(math.ceil(-0.5)) print(math.floor(2.2)) print(math.floor(-0.5)) Explanation:ceil(x) always rounds up.floor(x) always rounds down, even for negative numbers.Round Up to Specific Decimal PlacesIf we want to round up to a specified decimal place, you can combine multiplication, math.ceil(), and division. python import math def round_up(num, dec=0): mult = 10 ** dec return math.ceil(num * mult) / mult print(round_up(2.1)) print(round_up(2.23, 1)) print(round_up(2.543, 2)) print(round_up(22.45, -1)) print(round_up(2352, -2)) Output3.0 2.3 2.55 30.0 2400.0 Explanation:Shifts decimal right, applies ceil(), and shifts back.Works for positive and negative numbers.We can follow the diagram below to understand round up and round down. Round up to the right and down to the left. Rounding up always rounds a number to the right on the number line and rounding down always rounds a number to the left on the number line. Round Down to Specific Decimal PlacesSimilarly, for rounding down we can use floor() function instead of ceil(): python import math def round_down(num, dec=0): mult = 10 ** dec return math.floor(num * mult) / mult print(round_down(2.5)) print(round_down(2.48, 1)) print(round_down(-0.5)) Explanation: Same shifting logic as round up, but using floor().Round Numbers Using NumPyNumPy provides the np.round() function for rounding arrays or numbers efficiently. Python import numpy as np arr = np.array([1.234, 2.567, 3.789]) print("Nearest integer:", np.round(arr)) print("Two decimals:", np.round(arr, 2)) OutputNearest integer: [1. 3. 4.] Two decimals: [1.23 2.57 3.79] Explanation:np.round(arr) rounds to the nearest whole number.np.round(arr, n) rounds to n decimals.Round Numbers in Python using Rounding Bias conceptWhen rounding large datasets, we must avoid rounding bias. Python supports several rounding methods for that and most generally used methods are:1. Rounding Half UpAlways rounds .5 upward. python import math def round_half_up(num, dec=0): mult = 10 ** dec return math.floor(num * mult + 0.5) / mult print(round_half_up(1.28, 1)) print(round_half_up(-1.5)) print(round_half_up(-1.225, 2)) Explanation: Adds 0.5 before flooring to simulate "round half up."2. Rounding Half DownAlways rounds .5 downward. python import math def round_half_down(num, dec=0): mult = 10 ** dec return math.ceil(num * mult - 0.5) / mult print(round_half_down(2.5)) print(round_half_down(-2.5)) print(round_half_down(2.25, 1)) Explanation: Subtracts 0.5 before applying ceil() to simulate "round half down."Rounding Half to Even (Bankers' Rounding)Python's built-in round() uses round half to even strategy, which rounds .5 to the nearest even number to minimize bias.If n is positive and d >= 5, round upIf n is positive and d = 5, round downIf n is negative and d >= 5, round downIf n is negative and d < 5, round upAfter rounding as per the rules mentioned above, we can shift the decimal place back to the left. python from decimal import Decimal print(Decimal("0.1")) print(Decimal(0.1)) print(Decimal("1.65").quantize(Decimal("1.0"))) print(Decimal("1.675").quantize(Decimal("1.00"))) Output0.1 0.1000000000000000055511151231257827021181583404541015625 1.6 1.68 Explanation:Decimal offers precise control over decimal rounding.quantize() rounds to specified decimal places with minimal floating-point error.Related articles:NumPyDecimal moduleFloor Ceilround() function in Python DOWNLOAD 51 PYTHON PROGRAMS PDF FREE In this tutorial, I will explain how to round numbers in Python. As a data scientist working for a US-based company, I often encounter situations where I need to round decimal numbers to the nearest integer or a specific number of decimal places. Let's get in and explore the various methods to round numbers in Python. Python provides built-in functions and modules that make rounding numbers easy. Let us see some important methods to achieve this task. Python has a built-in round() function that takes two arguments: the number to be rounded and an optional number of decimal places. The basic syntax is as follows: rounded_number = round(number, ndigits) Here's an example: price = 9.99 rounded_price = round(price, 1) print(rounded_price) Output: 10.0 I have executed the above example and added the screenshot below. In this example, we have a variable price with a value of 9.99. We use the round() function to round it to one decimal place, which results in 10.0. If you leave the ndigits argument, round() will round the number to the nearest integer: score = 85.67 rounded_score = round(score) print(rounded_score) Output: 86 Here, the score of 85.67 is rounded to the nearest integer, which is 86. Read How to Check if a Number is Even or Odd in Python Let's say you're working on a project for a US-based e-commerce company, and you need to display product prices rounded to two decimal places. You can achieve this using the round() function: product_price = 19.9873 display_price = round(product_price, 2) print(display_price) Output: 19.99 I have executed the above example and added the screenshot below. In this case, the product_price is rounded to two decimal places, resulting in 19.99. Check out How to Generate Random 4-Digit Numbers in Python If you need to round a number up to the nearest integer, you can use the math.ceil() function from the math module. Here's an example: import math age = 25.4 rounded_age = math.ceil(age) print(rounded_age) Output: 26 I have executed the above example and added the screenshot below. In this example, the age 25.4 is rounded up to the nearest integer, which is 26. Read How to Check if a Python String Contains Only Numbers Similarly, if you want to round a number down to the nearest integer, you can use the math.floor() function: import math temperature = 98.7 rounded_temperature = math.floor(temperature) print(rounded_temperature) Output: 98 Here, the temperature of 98.7 is rounded down to the nearest integer, which is 98. Read How to Floor a Number in Python Python's decimal module provides more precise control over rounding behavior. It allows you to specify the rounding mode explicitly. Here's an example: from decimal import Decimal, ROUND_HALF_UP amount = Decimal('7.85') rounded_amount = amount.quantize(Decimal('0.1'), rounding=ROUND_HALF_UP) print(rounded_amount) Output: 7.9 In this example, we create an Decimal object with the value '7.85'. We then use the quantize() method to round it to one decimal place using the ROUND_HALF_UP rounding mode, which rounds numbers ending in 5 up to the next digit. Check out How to Find Factors of a Number in Python If you're working with NumPy arrays, you can use the numpy.round() function to round the elements of an array: import numpy as np data = np.array([1.23, 4.56, 7.89]) rounded_data = np.round(data, 1) print(rounded_data) Output: [1.2 4.6 7.9] Here, we have a NumPy array data containing decimal numbers. We use np.round() to round the elements of the array to one decimal place. Read How to Format Numbers as Currency in Python Suppose when you are working with floating-point numbers in Python. Due to the way floating-point numbers are represented internally, rounding can sometimes produce unexpected results. For example, let's say you're calculating the average temperature in New York City: temperatures = [75.4, 68.2, 80.9, 72.5] average_temperature = round(sum(temperatures) / len(temperatures), 1) print(average_temperature) Output: 74.2 Although the expected result is 74.3, the actual output is 74.2 due to the limitations of floating-point arithmetic. To mitigate such issues, you can use the decimal module for more precise decimal calculations: from decimal import Decimal temperatures = [Decimal('75.4'), Decimal('68.2'), Decimal('80.9'), Decimal('72.5')] average_temperature = round(sum(temperatures) / len(temperatures), 1) print(average_temperature) Output: 74.3 By using Decimal objects instead of floating-point numbers, you can achieve more accurate results. Check out How to Get the Decimal Part of a Number in Python In this tutorial, we explored various methods to round numbers in Python. We covered the built-in round() function, the math.ceil() and math.floor() functions for rounding up and down, the decimal module for rounding control, and rounding NumPy arrays using numpy.round(). You may also like to read: I am Bijay Kumar, a Microsoft MVP in SharePoint. Apart from SharePoint, I started working on Python, Machine learning, and artificial intelligence for the last 5 years. During this time I got expertise in various Python libraries also like Tkinter, Pandas, NumPy, Turtle, Django, Matplotlib, Tensorflow, Scipy, Scikit-Learn, etc… for various clients in the United States, Canada, the United Kingdom, Australia, New Zealand, etc. Check out my profile. When working with float values (numbers with decimal values) in our Python program, we might want to round them up or down, or to the nearest whole number. In this article, we'll see some built-in functionalities that let us round numbers in Python. And we'll see how to use them with some examples. We'll start with the round() function. By default, it rounds a number to the nearest whole number. We'll also see how to use the function's parameters to change the type of result returned to us. We'll then talk about the math.ceil() and math.floor() methods which rounds up and rounds down a number to the nearest whole number/integer respectively. These two methods are from the built-in math module in Python. The round() function takes in two parameters. Here's what the syntax looks like: round(number, decimal_digits) The first parameter – number – is the number we are rounding to the nearest whole number. The second parameter – decimal_digits – is the number of decimals to be returned. The default value is 0. Let's see some examples. x = 2.56789 print(round(x)) In our first example, we're using only one parameter – the number to be rounded, which is 2.56789. When we passed in the number variable to the round() function, it got rounded to the nearest whole number which is 3. That's how easy it is to use! Now, let's work with the second parameter. x = 2.56789 print(round(x, 2)) The code above is similar to the last example except for the second parameter. We passed in a value of two. This will round the number to the nearest hundredth (two decimal places). In our case, 2.57 was returned. That is, 2.56789 to 2.57. Let's see one last example to fully understand how the second parameter works. x = 2.56789 print(round(x, 3)) Now, we've made the second parameter 3. We'll get the number rounded to the nearest thousandth (three decimal places). The initial number – 2.56789 – was rounded to 2.568. The math.ceil() method simple takes in the number to be rounded up as its parameter. Here's what the syntax looks like: math.ceil(number) Here's an example: import math x = 5.57468465 print(math.ceil(x)) In the code above, you'll notice that we first imported the math module: import math. This give us access to all the methods provided by the module. We created an x variable which has 5.57468465 as its value. In order to round this number up to the nearest whole number, we passed in the number (in the x variable) to the math.ceil() method: math.ceil(x). The resulting value from this operation, as can be seen in the code above, is 6. Just like we did in the last section, in order to use the math.floor() method, we must first import the math module. Here's the syntax for math.floor() method: math.floor(number) Let's see an example. import math x = 5.57468465 print(math.floor(x)) As expected, we passed in the number to be rounded down to the math.floor() method: math.floor(x). The x variable has the number 5.57468465 stored in it. This number got rounded down to 5. In this article, we talked about three built-in functionalities in Python that let us round numbers. The round() function rounds a number to the nearest whole number. The math.ceil() method rounds a number up to the nearest whole number while the math.floor() method rounds a number down to the nearest whole number. These two methods are both accessible through the math module. With the examples given in each section, we were able to see how to use each functionality to obtain our desired result. Happy coding! This article will show you how to round numbers up, down, and to the nearest 1, 10 100 (or any number) in the Python programming language. Truncation vs. Rounding Truncation isn't technically rounding, but it serves a similar purpose, so I'll include it here. It's the simplest way to shorten a number – just cutting off the characters at the end until it's in the format you require. Rounding gets the nearest number, whereas truncation cuts off digits from the number. For example: To round 2.9 to the nearest whole number would give 3 To truncate 2.9 to the nearest whole number would give 2 Using The math.trunc() Method to Return the Truncated Integer Part of a Number The math.trunc() method is part of the Python math library and performs truncation to the nearest integer (whole number). Here it is in action: # Import the math library import math # print a truncated number print(math.trunc(3.7)) # Will print the number 3 As it is simply removing all information after the decimal place, it will round any negative number up and any positive number down. Truncation to a Set Number of Decimals Python lacks a built-in function for truncating to an arbitrary number of decimals (rather than just the nearest integer as above). This function fills this role – you can copy and paste it straight into your code. # Import the math library import math # define a function to truncate a number to a given number of decimal places (defaulting to 0 decimal places) def truncateToDecimals(number, decimals=0): if not isinstance(decimals, int): raise TypeError("decimals places must be an integer.") elif decimals < 0: raise ValueError("decimals must be greater than or equal to 0") if decimals == 0: return math.trunc(number) return math.trunc(number * 10.0 ** decimals) / 10.0 ** decimals Here it is in use: truncateToDecimals(6.76666, 3) # Will return 6.766 Rounding to A Specified Number of Decimal Places The Python round() function can be used to round a number to a set number of decimal places. Here it is in action: round(8.9873, 2) # Will return 8.99 The round() function accepts 2 parameters – the number to round followed by the number of decimal places to round to. If the number of decimal places is not supplied, it defaults to 0. Using math.ceil() to Round Up To The Nearest Integer The math.ceil() method rounds a given number up to the nearest integer. math.ceil(2.4) # Will return 3 math.ceil(-3.9) # will return -3 It gets the nearest integer that is greater than or equal to the number given to it. Using math.floor() to Round Down To The Nearest Integer Conversely, math.floor() rounds a given number down to the nearest integer: math.floor(2.4) # Will return 2 math.floor(-3.9) # Will return -4 It gets the nearest integer that is less than or equal to the number given to it. Rounding to the Nearest 2, 10, 5, Whatever The below functions will accept a number and round it to the nearest whole number in whatever base you want, defaulting to base 10: In Python 2 def roundToNearest(inputNumber, base=10): return int(base * round(float(inputNumber)/base)) In Python 3 def roundToNearest(inputNumber, base=10): return base * round(inputNumber/base) This can be useful if rounding to calculate change for a currency in which the smallest denomination coin is larger than the smallest unit of currency (for example, Australia's smallest value coin is 5 cents, but the currency allows for prices in 1 cent increments). Conclusion That pretty much covers the main rounding methods for Python. There are several popular math and data processing libraries for Python (like NumPy) – I'll dig into some of the functionality of those in future articles – be sure to subscribe to our Twitter to stay up to date. Meanwhile, now that you have some rounded numbers why not plot them on a scatter plot? Python is considered one of the easier programming languages to learn and boasts a massive standard library. Not to mention, it supplies the flexibility to approach problems in different ways. Case in point: rounding a number. There are a few different ways you can round a number in Python. In this post, we will look at four approaches, beginning with the built-in round() function. Rounding Numbers with the round() Function Using the round() function is one of the most straightforward ways to round a number in Python. It accepts two numeric arguments, "n" and "ndigits." It processes these inputs, rounding the number held in "n" to "ndigits." By default, the value of the "ndigits" argument is zero. So, if you don't give it any value, the number in "n" will turn into an integer. If you've handled number rounding before, there's a good chance that you approach it this way: You round the number "n" to "p" decimal places by shifting n's decimal point by p places. The value in n is multiplied by 10p to get a value "m." You then look at the number in the m's first decimal place. Let's say this number is "d." If "d" is less than 5, you round m down; else, you round m up. Lastly, you move the decimal point back p places by dividing m by 10p. The algorithm is quite straightforward. Even if you don't usually approach rounding this way, you should be able to understand it. All it's doing is taking the last digit after the decimal point and rounding down the number if it is below 5. If it is above 5, the algorithm round it up. Meaning if you try to round 2.5 with this algorithm, it'll round up to 3. But if you try to round 1.64 with it, it'll round down to 1.6. But this is not how the round() function works. Here's what happens when you try to round 2.5 with the function. >>> round(2.5) 2 Surprising, right? But what happens when you try to round 1.5 with the same function? >>> round(1.5) 2 Which isn't what you'd expect. But these outputs aren't a cause for alarm – you don't have to raise an issue on the Python bug tracker. This is because the method is designed to work this way. This is because Python uses the IEEE 754 standard for rounding. The approach is termed "banker's rounding," and the idea behind it is that the numbers are rounded to the nearest value with an even least significant digit. If you don't think this rounding strategy suits your needs, there are three other ways you can round a number in Python. Rounding Numbers by Truncating Truncating is the simplest way of rounding numbers but is also the most primitive. The idea behind truncating is that all the digits after a given position are replaced with zero. Value Truncated To Result 14. 568 Tens place 10 14. 568 Ones place 14 14. 568 Tenths place 14.5 14. 568 Hundredths place 14.56 Here's what the truncation algorithm does: The number you supply is multiplied by 1000. This shifts the decimal point three places to the right. The integer part of the new number is taken with int(). This number is divided by 1000 to shift the decimal places back three places. It's easy to generalize this process – all you have to do is replace "1000" with 10p, with "p" indicating the number of decimal places to truncate. Here's what the function will look like: def truncate(n, decimals=0): multiplier = 10 ** decimals return int(n * multiplier) / multiplier The second argument in this truncate() function defaults to 0. The function returns the integer part of the number passed to it if no second argument is passed. You can use this function equally well with the positive and negative numbers: >>> truncate(12.5) 12.0 >>> truncate(-5.963, 1) -5.9 >>> truncate(1.625, 2) 1.62 >>> truncate(125.6, -1) 120.0 >>> truncate(-1374.25, -3) -1000.0 You may have noticed that truncating a positive number results in rounding down. On the other hand, truncating a negative number results in it rounding up. If the program you're writing needs you to round up or round down numbers specifically, you can write programs for both of these applications. Let's explore what these rounding methods look like. Rounding Up Numbers The idea of rounding up a number involves rounding it to the next higher number up to a specific number of digits. Here's a table that summarizes this: Value Truncated To Result 14. 568 Tens place 20 14. 568 Ones place 15 14. 568 Tenths place 14.6 14. 568 Hundredths place 14.57 You can use the ceil() function in the math module to round up numbers in Python. As you might be able to guess, the name of the ceil() method is derived from the word "ceiling." The term ceiling in math describes the nearest integer that is greater than or equal to a given number. By the same token, the nearest integer that is lower than or equal to a given number is its floor. You must note that the numbers that lie between two integers are not themselves integers. More importantly, the ceiling of two intervals is the greater of the two intervals. Let's go over this concept with an example. The number 2.2 lies between 2 and 3, making its floor 2 and its ceiling 3. You might be aware of the concept of the ceiling function, which maps every number to its ceiling. The function also accepts integers, mapping their ceilings to the integer itself. Python also provides a ceiling function math.ceil(), which returns the nearest integer greater than or equal to the input. >>> import math >>> math.ceil(3.2) 4 >>> math.ceil(4) 4 >>> math.ceil(-0.7) 0 So, this is an excellent built-in Python function you can use to round up numbers. But you also have the option of writing your own function, like so: def round_up(n, decimals=0): multiplier = 10 ** decimals return math.ceil(n * multiplier) / multiplier It might seem like this round_up() function is a lot like the truncate() function. This is because their algorithms are quite similar. It shifts the decimal point in n the right number of places to the right. This is done by multiplying n by 10 ** decimals. The derived value is rounded up to the nearest integer with the math.ceil() function. Then, the program shifts the decimal point to the left by dividing the value by 10 ** decimals. Most techniques for rounding numbers have similar steps – the decimal point is shifted, some rounding method is applied, then the decimal point is shifted back. This pattern of rounding numbers is so popular because it's the most obvious mental algorithm humans came up with to round numbers. Here are a few instances of the round-up function in action: >>> round_up(2.1) 3.0 >>> round_up(4.23, 1) 4.3 >>> round_up(7.543, 2) 7.55 It's also worth noting that you can pass a negative value to the "decimals" part of the algorithm, as you would in truncate(). Here's how that would work: >>> round_up(32.45, -1) 40.0 >>> round_up(1452, -2) 1500 When a negative number is passed to the function in this way, the first argument written in the function rounds it to the appropriate number of digits to the left of the decimal point. Knowing this, does this make you curious about what passing a negative value to round_up() would do? Let's have a look: >>> round_up(-1.5) -1.0 You might have expected expecting symmetry around zero when rounding numbers. This might have led you to think that just like 1.5 is rounded to 2, -1.5 must round up to -2.0. You need to remember how the math.ceil() function works to make sense of the real result. It helps to remember the difference between rounding up and down. Visualize a number scale going from -5 to 5. Rounding up means the value jumps from left to right. In contrast, rounding down means the value jumps from right to left. Rounding Down Numbers Rounding down a number is the opposite of rounding it up. The idea of rounding down a number involves rounding it to the next lower number up to a specific number of digits. Here's a table that summarizes this: Value Truncated To Result 14. 568 Tens place 10 14. 568 Ones place 14 14. 568 Tenths place 14.5 14. 568 Hundredths place 14.56 Rounding a number down in Python involves more or less the same algorithm that round_up() and truncate() use. You shift the decimal point, round to an integer, and shift back the decimal point. The nice thing about Python is that the standard library comes with the math.floor() method, which does the opposite of what the ceil() method does. It rounds to the floor of the number in question after shifting the decimal point. >>> math.floor(2.2) 2 >>> math.floor(-1.5) -2 Let's now look at how you would write the round_down() function: def round_down(n, decimals=0): multiplier = 10 ** decimals return math.floor(n * multiplier) / multiplier So, as you can see, the program is the same as the round_up() program, except math.ceil() has been replaced with math.floor(). Let's try putting a few different values in round_down(): >>> round_down(2.5) 2 >>> round_down(3.37, 1) 3.3 >>> round_down(-0.7) -1 Writing a round_up() or round_down() function is an excellent way to round numbers quickly. But bear in mind that the impact on the data is extreme. If you use these functions in a large dataset, the dataset will likely become significantly less precise. This decrease in precision can dramatically alter computations done using the data. ❮ Built-in Functions Definition and Usage The round() function returns a floating point number that is a rounded version of the specified number, with the specified number of decimals. The default number of decimals is 0, meaning that the function will return the nearest integer. Syntax Parameter Values Parameter Description number Required. The number to be rounded digits Optional. The number of decimals to use when rounding the number. Default is 0 More Examples Round to the nearest integer: x = round(5.76543)print(x) Try it Yourself » ❮ Built-in Functions This article explains how to round numbers (floating point numbers float and integers int) in Python. For more information on rounding down and up decimals (floor and ceiling), refer to the following article. Round up/down decimals in Python (math.floor, math.ceil) For details on the round() function in NumPy and pandas, refer to the following articles. Built-in round() function In Python, the built-in round() function is available. Built-in Functions - round() — Python 3.12.1 documentation Specify the number to be rounded as the first argument and the number of decimal places to round to as the second argument, ndigits. Round decimals to a specific number of digits For floating point numbers (float), if the second argument is omitted, round() rounds to and returns an integer (int). f = 123.456 print(round(f)) # 123 print(type(round(f))) # When a second argument is specified, it returns a floating point number (float). Specifying a positive integer rounds to that number of decimal places, while specifying a negative integer rounds to that number of integer places. For example, -1 rounds to the tens place, -2 to the hundreds place. 0 rounds to an integer, but returns float. print(round(f, 1)) # 123.5 print(round(f, 2)) # 123.46 print(round(f, -1)) # 120.0 print(round(f, -2)) # 100.0 print(round(f, 0)) # 123.0 print(type(round(f, 0))) # Round integers to a specific number of digits For integers (int), round() returns the original number when the second argument is omitted, 0, or any positive integer. If a negative integer is specified, it rounds to the corresponding integer digit (-1 for tens, -2 for hundreds, etc.). In all cases, it returns an integer (int). i = 99518 print(round(i)) # 99518 print(round(i, 2)) # 99518 print(round(i, -1)) # 99520 print(round(i, -2)) # 99500 print(round(i, -3)) # 100000 round() rounds half to even (bankers' rounding) round() uses bankers' rounding, which means it rounds half to even. For example, 0.5 and 2.5 are rounded to 0 and 2, respectively. For the built-in types supporting round(), values are rounded to the closest multiple of 10 to the power minus ndigits; if two multiples are equally close, rounding is done toward the even choice (so, for example, both round(0.5) and round(-0.5) are 0, and round(1.5) is 2). Built-in Functions - round() — Python 3.12.1 documentation print('0.5 =>', round(0.5)) print('1.5 =>', round(1.5)) print('2.5 =>', round(2.5)) print('3.5 =>', round(3.5)) print('4.5 =>', round(4.5)) # 0.5 => 0 # 1.5 => 2 # 2.5 => 2 # 3.5 => 4 # 4.5 => 4 print(' 5 =>', round(5, -1)) print('15 =>', round(15, -1)) print('25 =>', round(25, -1)) print('35 =>', round(35, -1)) print('45 =>', round(45, -1)) # 5 => 0 # 15 => 20 # 25 => 20 # 35 => 40 # 45 => 40 Rounding to even occurs only when the fraction is exactly 0.5; for example, 2.5 is rounded to 2, but 2.51 is rounded to 3. print('2.49 =>', round(2.49)) print('2.50 =>', round(2.5)) print('2.51 =>', round(2.51)) # 2.49 => 2 # 2.50 => 2 # 2.51 => 3 Rounding beyond the first decimal place may not always conform to the definition of rounding half to even. print('0.05 =>', round(0.05, 1)) print('0.15 =>', round(0.15, 1)) print('0.25 =>', round(0.25, 1)) print('0.35 =>', round(0.35, 1)) print('0.45 =>', round(0.45, 1)) # 0.05 => 0.1 # 0.15 => 0.1 # 0.25 => 0.2 # 0.35 => 0.3 # 0.45 => 0.5 This inconsistency arises because, as mentioned in the documentation, floating point numbers cannot always represent decimals precisely. Note: The behavior of round() for floats can be surprising: for example, round(2.675, 2) gives 2.67 instead of the expected 2.68. This is not a bug: it's a result of the fact that most decimal fractions can't be represented exactly as a float. Built-in Functions - round() — Python 3.12.1 documentation For example, increasing the number of display digits shows that 0.15 is actually 0.14999.... Since the fraction is less than 0.05, it rounds to 0.1. How to use f-strings in Python print(f'{0.15:.20}') # 0.14999999999999999445 To accurately handle decimals or use different rounding modes, consider using the decimal module from the standard library, described next. decimal.quantize() The decimal module of the standard library allows for accurate decimal floating point arithmetic. The decimal module, being part of the standard library, requires no additional installation but must be imported to be used. In the following sample code, it is imported as shown. Create Decimal objects You can create Decimal objects with Decimal(). Specifying a floating point number (float) creates a Decimal with the actual value. print(Decimal(0.05)) # 0.05000000000000000277555756156289135105907917022705078125 print(type(Decimal(0.05))) # source: decimal_quantize.py This indicates that the floating point number 0.05 contains an error, which is why values such as 0.05 are rounded to unexpected values by the built-in round() function. 0.5 (= 1/2) and 0.25 (= 1/4) can be precisely represented. Specifying a string (str) creates a Decimal with exactly that value. To convert float to str, use str(). Round decimals to a specific number of digits The quantize() method of Decimal enables precise rounding with various modes. decimal.Decimal.quantize() — Python 3.12.1 documentation Specify a Decimal with the desired precision as the first argument. You can specify Decimal() with a string like '0.1' or '0.01'. To round the integer part, use scientific notation like '1E1'. More details will be discussed later. f = 123.456 print(Decimal(str(f)).quantize(Decimal('0'), ROUND_HALF_UP)) print(Decimal(str(f)).quantize(Decimal('0.1'), ROUND_HALF_UP)) print(Decimal(str(f)).quantize(Decimal('0.01'), ROUND_HALF_UP)) # 123 # 123.5 # 123.46 source: decimal_quantize.py Specify the rounding mode as the second argument, rounding. For example, 0.5 rounds to 1 with ROUND_HALF_UP. print('0.4 =>', Decimal(str(0.4)).quantize(Decimal('0'), ROUND_HALF_UP)) print('0.5 =>', Decimal(str(0.5)).quantize(Decimal('0'), ROUND_HALF_UP)) print('0.6 =>', Decimal(str(0.6)).quantize(Decimal('0'), ROUND_HALF_UP)) # 0.4 => 0 # 0.5 => 1 # 0.6 => 1 source: decimal_quantize.py ROUND_HALF_EVEN performs the same even rounding as the built-in round() function. By specifying values as strings in Decimal(), more accurate representation is ensured, avoiding the floating point errors typically associated with round(), and thereby producing the expected result. print('0.05 =>', Decimal(str(0.05)).quantize(Decimal('0.1'), ROUND_HALF_EVEN)) print('0.15 =>', Decimal(str(0.15)).quantize(Decimal('0.1'), ROUND_HALF_EVEN)) print('0.25 =>', Decimal(str(0.25)).quantize(Decimal('0.1'), ROUND_HALF_EVEN)) print('0.35 =>', Decimal(str(0.35)).quantize(Decimal('0.1'), ROUND_HALF_EVEN)) print('0.45 =>', Decimal(str(0.45)).quantize(Decimal('0.1'), ROUND_HALF_EVEN)) # 0.05 => 0.0 # 0.15 => 0.2 # 0.25 => 0.2 # 0.35 => 0.4 # 0.45 => 0.4 source: decimal_quantize.py For a list of rounding modes, refer to the official documentation. There are various rounding modes available besides ROUND_HALF_UP and ROUND_HALF_EVEN. decimal - Rounding modes — Python 3.12.1 documentation The quantize() method returns a Decimal, which can be converted to a float using float(). However, this conversion results in a value that is limited to the representational capabilities of a float. d = Decimal('123.456').quantize(Decimal('0.01'), ROUND_HALF_UP) print(d) # 123.46 print(type(d)) # f = float(d) print(f) # 123.46 print(type(f)) # print(Decimal(f)) # 123.4599999999999937472239253111183643341064453125 print(Decimal(str(f))) # 123.46 source: decimal_quantize.py Round integers to a specific number of digits When using the quantize() method to round to integer digits, specifying something like '10' as the first argument does not return the desired result. i = 99518 print(Decimal(i).quantize(Decimal('10'), ROUND_HALF_UP)) # 99518 source: decimal_quantize.py This is because quantize() performs rounding based on the exponent (exponent) of the Decimal object. It is necessary to use a scientific notation string (for example, '1E1'). You can check exponent with the as_tuple() method. print(Decimal('10').as_tuple()) # DecimalTuple(sign=0, digits=(1, 0), exponent=0) print(Decimal('1E1').as_tuple()) # DecimalTuple(sign=0, digits=(1,), exponent=1) source: decimal_quantize.py Since integers can be accurately represented as int types, you can specify them directly as integers (int) in Decimal(). Of course, it may be specified as a string. print(Decimal(i).quantize(Decimal('1E1'), ROUND_HALF_UP)) print(Decimal(i).quantize(Decimal('1E2'), ROUND_HALF_UP)) print(Decimal(i).quantize(Decimal('1E3'), ROUND_HALF_UP)) # 9.952E+4 # 9.95E+4 # 1.00E+5 source: decimal_quantize.py If you want to use regular notation instead of scientific notation, use the function introduced in the official documentation. decimal - FAQ — Python 3.12.1 documentation def remove_exponent(d): return d.quantize(Decimal(1)) if d == d.to_integral() else d.normalize() d = Decimal(i).quantize(Decimal('1E2'), ROUND_HALF_UP) print(d) # 9.95E+4 d_remove = remove_exponent(d) print(d_remove) # 99500 print(type(d_remove)) # source: decimal_quantize.py To convert Decimal to an integer (int), use int(). Custom functions for standard rounding (rounding half up) Although the decimal module is often the better choice due to its precision, if you simply want to perform standard rounding (rounding half up), defining a custom function is another option. For example, define the following function. The return value is always a floating point number (float). def my_round(number, ndigits=0): p = 10**ndigits return (number * p * 2 + 1) // 2 / p If you don't need to specify the number of digits and simply want to round to an integer, a more concise approach can be adopted. def my_round_int(number): return int((number * 2 + 1) // 2) Round decimals to a specific number of digits Here is an example of the result using the function defined above. my_round() always returns a floating point number (float), so use int() if you want to convert it to an integer (int). f = 123.456 print(my_round(f)) # 123.0 print(int(my_round(f))) # 123 print(my_round(f, 1)) # 123.5 print(my_round(f, 2)) # 123.46 print(my_round_int(f)) # 123 Unlike the built-in round() function, this rounds 0.5 to 1. print('0.4 =>', int(my_round(0.4))) print('0.5 =>', int(my_round(0.5))) print('0.6 =>', int(my_round(0.6))) # 0.4 => 0 # 0.5 => 1 # 0.6 => 1 Round integers to a specific number of digits Here is an example of the result using the function defined above. i = 99518 print(int(my_round(i, -1))) # 99520 print(int(my_round(i, -2))) # 99500 print(int(my_round(i, -3))) # 100000 Unlike the built-in round() function, this rounds 5 to 10. print('4 =>', int(my_round(4, -1))) print('5 =>', int(my_round(5, -1))) print('6 =>', int(my_round(6, -1))) # 4 => 0 # 5 => 10 # 6 => 10 Note: In the case of negative values With the function defined above, -0.5 is rounded to 0. print('-0.4 =>', int(my_round(-0.4))) print('-0.5 =>', int(my_round(-0.5))) print('-0.6 =>', int(my_round(-0.6))) # -0.4 => 0 # -0.5 => 0 # -0.6 => -1 There are various approaches to round negative values. If you want to round -0.5 to -1, for example, do the following. Convert the input value to an absolute value using the built-in abs() function, and multiply by the sign obtained using math.copysign() at the end. import math def my_round2(number, ndigits=0): p = 10**ndigits return (abs(number) * p * 2 + 1) // 2 / p * math.copysign(1, number) print('-0.4 =>', int(my_round2(-0.4))) print('-0.5 =>', int(my_round2(-0.5))) print('-0.6 =>', int(my_round2(-0.6))) # -0.4 => 0 # -0.5 => -1 # -0.6 => -1 Make sure to click on the correct subheading depending on how you need to round the number.Use the round() function to round a float to the nearest 10th (0.1), e.g. result = round(4.5678, 1).The round() function will return the number rounded to 1-digit precision after the decimal point.main.pyCopied!# □ round a float to the nearest 10th (0.1) result_1 = round(4.5678, 1) print(result_1) # □ 4.6 result_2 = round(4.1234, 1) print(result_2) # □ 4.1 # -- # □ print a float rounded to the nearest 10th (0.1) my_float = 4.5678 my_str_1 = f'{my_float:.1f}' print(my_str_1) # □ '4.6' my_str_2 = f'{my_float:.2f}' print(my_str_2) # □ '4.57' The code for this article is available on GitHubThe round() function takes the following 2 parameters:NameDescriptionnumberthe number to round to ndigits precision after the decimalndigitsthe number of digits after the decimal, the number should have after the operation (optional)The round function returns the number rounded to ndigits precision after the decimal point.If ndigits is omitted, the function returns the nearest integer.main.pyCopied!my_num = 3.456 result_1 = round(my_num) print(result_1) # □ 3 result_2 = round(my_num, 1) print(result_2) # □ 3.5 If you need to print a floating-point number rounded to the nearest 10th (0.1), use a formatted string literal.main.pyCopied!my_float = 4.5678 my_str_1 = f'{my_float:.1f}' print(my_str_1) # □ '4.6' my_str_2 = f'{my_float:.2f}' print(my_str_2) # □ '4.57' Formatted string literals (f-strings) let us include expressions inside of a string by prefixing the string with f.main.pyCopied!my_str = 'is subscribed:' my_bool = True result = f'{my_str} {my_bool}' print(result) # □ is subscribed: True Make sure to wrap expressions in curly braces - {expression}.We are also able to use the format specification mini-language in expressions in f-strings.main.pyCopied!my_float = 1.45678 result_1 = f'{my_float:.1f}' print(result_1) # □ '1.5' result_2 = f'{my_float:.2f}' print(result_2) # □ '1.46' result_3 = f'{my_float:.3f}' print(result_3) # □ '1.457' The f character between the curly braces stands for fixed-point notation.The character formats the number as a decimal number with the specified number of digits following the decimal point.# Table of Contents# Round a float to the nearest 0.5 in PythonTo round a float to the nearest 0.5:Call the round() function passing it the number multiplied by 2.Divide the result by 2.The result of the calculation is the number rounded to the nearest 0.5.main.pyCopied!import math # □ Round number to nearest 0.5 def round_to_nearest_half_int(num): return round(num * 2) / 2 print(round_to_nearest_half_int(3.1)) # □ 3.0 print(round_to_nearest_half_int(3.7)) # □ 3.5 # -- # □ Round number UP to nearest 0.5 def round_up_to_nearest_half_int(num): return math.ceil(num * 2) / 2 print(round_up_to_nearest_half_int(3.1)) # □ 3.5 print(round_up_to_nearest_half_int(3.7)) # □ 4.0 # -- # □ Round number DOWN to nearest 0.5 def round_down_to_nearest_half_int(num): return math.floor(num * 2) / 2 print(round_down_to_nearest_half_int(3.9)) # □ 3.5 print(round_down_to_nearest_half_int(3.4)) # □ 3.0 The code for this article is available on GitHubWe used the round() function to round a number to the nearest 0.5.When passed a single argument, the round() function rounds to the nearest integer.main.pyCopied!print(round(7.4)) # □ 7 print(round(7.6)) # □ 8 Here is a step-by-step example of rounding a number up to the nearest 0.5.main.pyCopied!print(3.1 * 2) # □ 6.2 print(3.7 * 2) # □ 7.4 print(round(3.1 * 2)) # □ 6 print(round(3.7 * 2)) # □ 7 print(round(3.1 * 2) / 2) # □ 3.0 print(round(3.7 * 2) / 2) # □ 3.5 This is a two-step process:Multiply the number by 2 and round the result to the nearest integer.Divide the result by 2 to get the number rounded to the nearest 0.5.# Round a float Up to the nearest 0.5 in PythonUse the math.ceil() method if you need to round a float up to the nearest 0.5.main.pyCopied!import math def round_up_to_nearest_half_int(num): return math.ceil(num * 2) / 2 print(round_up_to_nearest_half_int(3.1)) # □ 3.5 print(round_up_to_nearest_half_int(3.7)) # □ 4.0 print(round_up_to_nearest_half_int(16.2)) # □ 16.5 The math.ceil method returns the smallest integer greater than or equal to the provided number.main.pyCopied!import math print(math.ceil(3.1)) # □ 4 print(math.ceil(3.9)) # □ 4 If the passed-in number has a fractional part, the math.ceil method rounds the number up.Here is a step-by-step example of rounding a number up to the nearest 0.5.main.pyCopied!import math print(6.1 * 2) # □ 12.2 print(6.6 * 2) # □ 13.2 print(math.ceil(6.1 * 2)) # □ 13 print(math.ceil(6.6 * 2)) # □ 14 print(math.ceil(6.1 * 2) / 2) # □ 6.5 print(math.ceil(6.6 * 2) / 2) # □ 7.0 This is a two-step process:Multiply the number by 2 and round the result up to the nearest integer.Divide the result by 2 to get the number rounded up to the nearest 0.5.# Round a float Down to the nearest 0.5 in PythonUse the math.floor() method to round a number down to the nearest 0.5.main.pyCopied!import math def round_down_to_nearest_half_int(num): return math.floor(num * 2) / 2 print(round_down_to_nearest_half_int(3.9)) # □ 3.5 print(round_down_to_nearest_half_int(3.4)) # □ 3.0 The math.floor method returns the largest integer less than or equal to the provided number.main.pyCopied!import math print(math.floor(3.9)) # □ 3 print(math.floor(3.1)) # □ 3 If the passed-in number has a fractional part, the math.floor method rounds the number down.Here is a step-by-step example of rounding a number down to the nearest 0.5.main.pyCopied!import math print(5.9 * 2) # □ 11.8 print(5.1 * 2) # □ 10.2 print(math.floor(5.9 * 2)) # □ 11 print(math.floor(5.1 * 2)) # □ 10 print(math.floor(5.9 * 2) / 2) # □ 5.5 print(math.floor(5.1 * 2) / 2) # □ 5.0 This is a two-step process:Multiply the number by 2 and round the result down to the nearest integer.Divide the result by 2 to get the number rounded down to the nearest 0.5.# Table of Contents# Round a number to the nearest 5 in PythonTo round a number to the nearest 5:Call the round() function passing it the number divided by 5.Multiply the result by 5.The result of the calculation is the number rounded to the nearest 5.main.pyCopied!import math # □ Round number to nearest 5 def round_to_nearest_5(num): return round(num / 5) * 5 print(round_to_nearest_5(2)) # □ 0 print(round_to_nearest_5(8)) # □ 10 print(round_to_nearest_5(26)) # □ 25 # -- # □ Round number UP to nearest 5 def round_up_to_nearest_5(num): return math.ceil(num / 5) * 5 print(round_up_to_nearest_5(23)) # □ 25 print(round_up_to_nearest_5(57)) # □ 60 # -- # □ Round number DOWN to nearest 5 def round_down_to_nearest_5(num): return math.floor(num / 5) * 5 print(round_down_to_nearest_5(121)) # □ 120 print(round_down_to_nearest_5(129)) # □ 125 The code for this article is available on GitHubWe used the round() function to round a number to the nearest 5.When passed a single argument, the round function rounds to the nearest integer.main.pyCopied!print(round(14.4)) # □ 14 print(round(14.6)) # □ 15 Here is a step-by-step example of rounding a number up to the nearest five.main.pyCopied!print(24 / 5) # □ 4.8 print(38 / 5) # □ 7.6 print(round(24 / 5)) # □ 5 print(round(38 / 5)) # □ 8 print(round(24 / 5) * 5) # □ 25 print(round(38 / 5) * 5) # □ 40 This is a two-step process:Divide the number by 5 and round the result to the nearest integer.Multiply the result by 5 to get the number rounded to the nearest 5.# Round a number Up to the nearest 5 in PythonUse the math.ceil() method to round a number up to the nearest 5.main.pyCopied!import math def round_up_to_nearest_5(num): return math.ceil(num / 5) * 5 print(round_up_to_nearest_5(23)) # □ 25 print(round_up_to_nearest_5(57)) # □ 60 The code for this article is available on GitHubThe math.ceil() method returns the smallest integer greater than or equal to the provided number.main.pyCopied!import math print(math.ceil(6.01)) # □ 7 print(math.ceil(6.99)) # □ 7 If the passed-in number has a fractional part, the math.ceil method rounds the number up.Here is a step-by-step example of rounding a number up to the nearest five.main.pyCopied!import math print(142 / 5) # □ 28.4 print(148 / 5) # □ 29.6 print(math.ceil(142 / 5)) # □ 29 print(math.ceil(148 / 5)) # □ 30 print(math.ceil(142 / 5) * 5) # □ 145 print(math.ceil(148 / 5) * 5) # □ 150 This is a two-step process:Divide the number by 5 and round the result up to the nearest integer.Multiply the result by 5 to get the number rounded up to the nearest 5.# Round a Number Down to the nearest 5 in PythonUse math.floor() if you need to round down to the nearest 5.main.pyCopied!import math def round_down_to_nearest_5(num): return math.floor(num / 5) * 5 print(round_down_to_nearest_5(121)) # □ 120 print(round_down_to_nearest_5(129)) # □ 125 The code for this article is available on GitHubThe math.floor method returns the largest integer less than or equal to the provided number.main.pyCopied!import math print(math.floor(3.99)) # □ 3 print(math.floor(3.01)) # □ 3 If the passed-in number has a fractional part, the math.floor method rounds the number down.Here is a step-by-step example of rounding a number down to the nearest 5.main.pyCopied!import math print(49 / 5) # □ 9.8 print(56 / 5) # □ 11.2 print(math.floor(49 / 5)) # □ 9 print(math.floor(56 / 5)) # □ 11 print(math.floor(49 / 5) * 5) # □ 45 print(math.floor(56 / 5) * 5) # □ 55 This is a two-step process:Divide the number by 5 and round the result down to the nearest integer.Multiply the result by 5 to get the number rounded down to the nearest 5.# Table of Contents# Round a Number to the nearest 10 in PythonUse the round() function to round a number to the nearest 10.main.pyCopied!import math # Round number to nearest 10 num_1 = 6 result_1 = round(num_1, -1) print(result_1) # □ 10 num_2 = 4 result_2 = round(num_2, -1) print(result_2) # □ 0 # -- # □ Round number UP to nearest 10 def round_up_to_nearest_10(num): return math.ceil(num / 10) * 10 print(round_up_to_nearest_10(3)) # □ 10 print(round_up_to_nearest_10(1)) # □ 10 # -- # □ Round number DOWN to nearest 10 def round_down_to_nearest_10(num): return math.floor(num / 10) * 10 print(round_down_to_nearest_10(19)) # □ 10 print(round_down_to_nearest_10(27)) # □ 20 The code for this article is available on GitHubWe used the round() function to round a number to the nearest 10.The round() function takes the following 2 parameters:NameDescriptionnumberthe number to round to ndigits precision after the decimalndigitsthe number of digits after the decimal, the number should have after the operation (optional)When ndigits is a negative number, the round() function rounds to the left of the decimal.If ndigits is -1, it rounds to the closest multiple of 10. When ndigits is -2, the function rounds to the nearest 100, etc.main.pyCopied!print(round(157, -1)) # □ 160 print(round(157, -2)) # □ 200 # Round a Number Up to the nearest 10 in PythonUse the math.ceil() method to round up to the nearest 10.main.pyCopied!import math def round_up_to_nearest_10(num): return math.ceil(num / 10) * 10 print(round_up_to_nearest_10(3)) # □ 10 print(round_up_to_nearest_10(1)) # □ 10 print(round_up_to_nearest_10(21)) # □ 30 The math.ceil method returns the smallest integer greater than or equal to the provided number.main.pyCopied!import math print(math.ceil(5.001)) # □ 6 print(math.ceil(5.99)) # □ 6 If the passed-in number has a fractional part, the math.ceil method rounds the number up.main.pyCopied!import math my_num = math.ceil(3 / 10) # □ 1 print(my_num) # □ 1 result = my_num * 10 print(result) # □ 10 Here is a step-by-step example of rounding a number up to the nearest 10.main.pyCopied!import math print(21 / 10) # □ 2.1 print(40 / 10) # □ 4.0 print(math.ceil(21 / 10)) # □ 3 print(math.ceil(40 / 10)) # □ 4 print(math.ceil(21 / 10) * 10) # □ 30 print(math.ceil(40 / 10) * 10) # □ 40 We first divide the number by 10 and then multiply with 10 to shift 1 decimal place to the right and left, so that math.ceil() works on the tens.This is a two-step process:Divide the number by 10 and round the result up to the nearest integer.Multiply the result by 10 to get the number rounded up to the nearest 10.# Round a Number Down to the nearest 10 in PythonUse the math.floor() method to round down to the nearest 10.main.pyCopied!import math def round_down_to_nearest_10(num): return math.floor(num / 10) * 10 print(round_down_to_nearest_10(19)) # □ 10 print(round_down_to_nearest_10(27)) # □ 20 print(round_down_to_nearest_10(42)) # □ 40 The math.floor method returns the largest integer less than or equal to the provided number.main.pyCopied!import math print(math.floor(3.999)) # □ 3 print(math.floor(3.001)) # □ 3 If the passed-in number has a fractional part, the math.floor method rounds the number down.Here is a step-by-step example of rounding a number down to the nearest 10.main.pyCopied!import math print(34 / 10) # □ 3.4 print(50 / 10) # □ 5.0 print(math.floor(34 / 10)) # □ 3 print(math.floor(50 / 10)) # □ 5 print(math.floor(34 / 10) * 10) # □ 30 print(math.floor(50 / 10) * 10) # □ 50 We first divide the number by 10 and then multiply with 10 to shift 1 decimal place to the right and left, so that math.floor() works on the tens.This is a two-step process:Divide the number by 10 and round the result down to the nearest integer.Multiply the result by 10 to get the number rounded down to the nearest 10.# Table of Contents# Round a number to the nearest 100 in PythonUse the round() function if you need to round a number to the nearest 100.main.pyCopied!import math # □ Round number to nearest 100 num_1 = 237 result_1 = round(num_1, -2) print(result_1) # □ 200 num_2 = 278 result_2 = round(num_2, -2) print(result_2) # □ 300 # -- # □ Round number UP to nearest 100 def round_up_to_nearest_100(num): return math.ceil(num / 100) * 100 print(round_up_to_nearest_100(311)) # □ 400 print(round_up_to_nearest_100(1)) # □ 100 # -- # □ Round number DOWN to nearest 100 def round_down_to_nearest_100(num): return math.floor(num / 100) * 100 print(round_down_to_nearest_100(546)) # □ 500 print(round_down_to_nearest_100(599)) # □ 500 The code for this article is available on GitHubWe used the round() function to round a number to the nearest 100.The round() function takes the following 2 parameters:NameDescriptionnumberthe number to round to ndigits precision after the decimalndigitsthe number of digits after the decimal, the number should have after the operation (optional)When ndigits is a negative number, the round() function rounds to the left of the decimal.If ndigits is -1, it rounds to the closest multiple of 10. When ndigits is -2, the function rounds to the nearest 100, etc.main.pyCopied!print(round(346, -1)) # □ 350 print(round(346, -2)) # □ 300 # Round a number Up to the nearest 100 in PythonUse the math.ceil() method if you need to round a number up to the nearest 100.main.pyCopied!import math def round_up_to_nearest_100(num): return math.ceil(num / 100) * 100 print(round_up_to_nearest_100(311)) # □ 400 print(round_up_to_nearest_100(1)) # □ 100 The math.ceil method returns the smallest integer greater than or equal to the provided number.main.pyCopied!import math print(math.ceil(123.001)) # □ 124 print(math.ceil(123.999)) # □ 124 If the passed-in number has a fractional part, the math.ceil method rounds the number up.Here is a step-by-step example of rounding a number up to the nearest hundred.main.pyCopied!import math print(346 / 100) # □ 3.46 print(600 / 100) # □ 6.0 print(math.ceil(346 / 100)) # □ 4 print(math.ceil(600 / 100)) # □ 6 print(math.ceil(346 / 100) * 100) # □ 400 print(math.ceil(600 / 100) * 100) # □ 600 We first divide the number by 100 and then multiply with 100 to shift 2 decimal places to the right and left, so that math.ceil() works on the hundreds.This is a two-step process:Divide the number by 100 and round the result up to the nearest integer.Multiply the result by 100 to get the number rounded up to the nearest 100.# Round a Number Down to the nearest 100 in PythonUse the math.floor() method to round a number down to the nearest 100.main.pyCopied!import math def round_down_to_nearest_100(num): return math.floor(num / 100) * 100 print(round_down_to_nearest_100(546)) # □ 500 print(round_down_to_nearest_100(599)) # □ 500 print(round_down_to_nearest_100(775)) # □ 700 The math.floor method returns the largest integer less than or equal to the provided number.main.pyCopied!import math print(math.floor(15.001)) # □ 15 print(math.floor(15.999)) # □ 15 If the passed-in number has a fractional part, the math.floor method rounds the number down.Here is a step-by-step example of rounding a number down to the nearest 100.main.pyCopied!print(488 / 100) # □ 4.88 print(251 / 100) # □ 2.51 print(math.floor(488 / 100)) # □ 4 print(math.floor(251 / 100)) # □ 2 print(math.floor(488 / 100) * 100) # □ 400 print(math.floor(251 / 100) * 100) # □ 200 We first divide the number by 100 and then multiply with 100 to shift 2 decimal places to the right and left, so that math.floor() works on the hundreds.This is a two-step process:Divide the number by 100 and round the result down to the nearest integer.Multiply the result by 100 to get the number rounded down to the nearest 100.# Table of Contents# Round a number to the nearest 500 in PythonUse the round() function to round a number to the nearest 500.main.pyCopied!import math # Round number to nearest 500 def round_to_nearest_500(num): return round(num / 500) * 500 print(round_to_nearest_500(777)) # □ 1000 print(round_to_nearest_500(1)) # □ 0 print(round_to_nearest_500(1400)) # □ 1500 # -- # □ Round number UP to nearest 500 def round_up_to_nearest_500(num): return math.ceil(num / 500) * 500 print(round_up_to_nearest_500(640)) # □ 1000 print(round_up_to_nearest_500(1)) # □ 500 # -- # □ Round number DOWN to nearest 500 def round_down_to_nearest_500(num): return math.floor(num / 500) * 500 print(round_down_to_nearest_500(999)) # □ 500 print(round_down_to_nearest_500(1840)) # □ 1500 The code for this article is available on GitHubWe used the round() function to round a number to

the nearest 500.When passed a single argument, the round function rounds to the nearest integer.main.pyCopied!print(round(13.4)) # 🔴 13 print(round(13.6)) # 🔴 14 Here is a step-by-step example of rounding a number up to the nearest five hundred.main.pyCopied!print(1750 / 500) # 🔴 3.5 print(1400 / 500) # 🔴 2.8 print(round(1750 / 500)) # 🔴 4 print(round(1400 / 500)) # 🔴 3 print(round(1750 / 500) * 500) # 🔴 2000 print(round(1400 / 500) * 500) # 🔴 1500 This is a two-step process:Divide the number by 500 and round the result to the nearest integer.Multiply the result by 500 to get the number rounded to the nearest 500.# Round a number Up to the nearest 500 in PythonUse the math.ceil() method to round a number up to the nearest 500.main.pyCopied!import math def round_up_to_nearest_500(num): return math.ceil(num / 500) * 500 print(round_up_to_nearest_500(640)) # 🔴 1000 print(round_up_to_nearest_500(1)) # 🔴 500 The math.ceil method returns the smallest integer greater than or equal to the provided number.main.pyCopied!import math print(math.ceil(456.001)) # 🔴 457 print(math.ceil(456.999)) # 🔴 457 If the passed-in number has a fractional part, the math.ceil method rounds the number up.Here is a step-by-step example of rounding a number up to the nearest five hundred.main.pyCopied!import math print(1346 / 500) # 🔴 2.692 print(1600 / 500) # 🔴 3.2 print(math.ceil(1346 / 500)) # 🔴 3 print(math.ceil(1600 / 500)) # 🔴 4 print(math.ceil(1346 / 500) * 500) # 🔴 1500 print(math.ceil(1600 / 500) * 500) # 🔴 2000 This is a two-step process:Divide the number by 500 and round the result up to the nearest integer.Multiply the result by 500 to get the number rounded up to the nearest 500.# Round a Number Down to the nearest 500 in PythonUse the math.floor() method to round a number down to the nearest 500.main.pyCopied!import math def round_down_to_nearest_500(num): return math.floor(num / 500) * 500 print(round_down_to_nearest_500(999)) # 🔴 500 print(round_down_to_nearest_500(1840)) # 🔴 1500 print(round_down_to_nearest_500(2840)) # 🔴 2500 The math.floor method returns the largest integer less than or equal to the provided number.main.pyCopied!import math print(math.floor(25.999)) # 🔴 25 print(math.floor(25.001)) # 🔴 25 If the passed-in number has a fractional part, the math.floor method rounds the number down.Here is a step-by-step example of rounding a number down to the nearest 500.main.pyCopied!import math print(4880 / 500) # 🔴 9.76 print(2510 / 500) # 🔴 5.02 print(math.floor(4880 / 500)) # 🔴 9 print(math.floor(2510 / 500)) # 🔴 5 print(math.floor(4880 / 500) * 500) # 🔴 4500 print(math.floor(2510 / 500) * 500) # 🔴 2500 This is a two-step process:Divide the number by 500 and round the result down to the nearest integer.Multiply the result by 500 to get the number rounded down to the nearest 500.# Table of Contents# Round a number to the nearest 1000 in PythonUse the round() function to round a number to the nearest 1000.main.pyCopied!import math # Round number to nearest 1000 num_1 = 4678 result_1 = round(num_1, -3) print(result_1) # 🔴 5000 num_2 = 4432 result_2 = round(num_2, -3) print(result_2) # 🔴 4000 # ---------- # 🔴 Round number UP to nearest 1000 def round_up_to_nearest_1000(num): return math.ceil(num / 1000) * 1000 print(round_up_to_nearest_1000(3100)) # 🔴 4000 print(round_up_to_nearest_1000(1)) # 🔴 1000 # ---------- # 🔴 Round number DOWN to nearest 1000 def round_down_to_nearest_1000(num): return math.floor(num / 1000) * 1000 print(round_down_to_nearest_1000(5999)) # 🔴 5000 print(round_down_to_nearest_1000(5004)) # 🔴 5000 The code for this article is available on GitHubIf you need to round a number to the nearest 500, scroll down to the relevant subheading.We used the round() function to round a number to the nearest 1000.The round function takes the following 2 parameters:NameDescriptionnumberthe number to round to ndigits precision after the decimalndigitsthe number of digits after the decimal, the number should have after the operation (optional)When ndigits is a negative number, the round() function rounds to the left of the decimal.If ndigits is -1, the function rounds to the closest multiple of 10.If ndigits is -2, it rounds to the nearest 100.If ndigits is -3, it rounds to the nearest 1000, etc.main.pyCopied!print(round(3456, -1)) # 🔴 3460 print(round(3456, -2)) # 🔴 3500 print(round(3456, -3)) # 🔴 3000 # Round a number Up to the nearest 1000 in PythonUse the math.ceil() method to round a number up to the nearest 1000.main.pyCopied!import math def round_up_to_nearest_1000(num): return math.ceil(num / 1000) * 1000 print(round_up_to_nearest_1000(3100)) # 🔴 4000 print(round_up_to_nearest_1000(1)) # 🔴 1000 print(round_up_to_nearest_1000(2350)) # 🔴 3000 The math.ceil method returns the smallest integer greater than or equal to the provided number.main.pyCopied!import math print(math.ceil(1234.001)) # 🔴 1235 print(math.ceil(1234.999)) # 🔴 1235 If the passed-in number has a fractional part, the math.ceil method rounds the number up.Here is a step-by-step example of rounding a number up to the nearest thousand.main.pyCopied!import math print(4258 / 1000) # 🔴 4.258 print(5600 / 1000) # 🔴 5.6 print(math.ceil(4258 / 1000)) # 🔴 5 print(math.ceil(5600 / 1000)) # 🔴 6 print(math.ceil(4258 / 1000) * 1000) # 🔴 5000 print(math.ceil(5600 / 1000) * 1000) # 🔴 6000 We first divide the number by 1000 and then multiply with 1000 to shift 3 decimal places to the right and left, so that math.ceil() works on the thousands.This is a two-step process:Divide the number by 1000 and round the result up to the nearest integer.Multiply the result by 1000 to get the number rounded up to the nearest 1000.# Round a Number Down to the nearest 1000 in PythonUse the math.floor() method to round a number down to the nearest 1000.main.pyCopied!import math def round_down_to_nearest_1000(num): return math.floor(num / 1000) * 1000 print(round_down_to_nearest_1000(5999)) # 🔴 5000 print(round_down_to_nearest_1000(5004)) # 🔴 5000 print(round_down_to_nearest_1000(7900)) # 🔴 7000 The math.floor method returns the largest integer less than or equal to the provided number.main.pyCopied!import math print(math.floor(13.999)) # 🔴 13 print(math.floor(13.001)) # 🔴 13 If the passed-in number has a fractional part, the math.floor method rounds the number down.Here is a step-by-step example of rounding a number down to the nearest 1000.main.pyCopied!import math print(5900 / 1000) # 🔴 5.9 print(4300 / 1000) # 🔴 4.3 print(math.floor(5900 / 1000)) # 🔴 5 print(math.floor(4300 / 1000)) # 🔴 4 print(math.floor(5900 / 1000) * 1000) # 🔴 5000 print(math.floor(4300 / 1000) * 1000) # 🔴 4000 The code for this article is available on GitHubWe first divide the number by 1000 and then multiply with 1000 to shift 3 decimal places to the right and left, so that math.floor() works on the thousands.This is a two-step process:Divide the number by 1000 and round the result down to the nearest integer.Multiply the result by 1000 to get the number rounded down to the nearest 1000.# Round a number to the nearest even number in PythonUse the round() function to round a number to the nearest even number.main.pyCopied!import math # 🔴 round number to nearest even number def round_to_nearest_even_number(num): return round(num / 2) * 2 print(round_to_nearest_even_number(3.1)) # 🔴 4 print(round_to_nearest_even_number(8.6)) # 🔴 8 # ---------- # 🔴 round a number UP to the nearest even number def round_up_to_nearest_even_number(num): return math.ceil(num / 2) * 2 print(round_up_to_nearest_even_number(3.1)) # 🔴 4 print(round_up_to_nearest_even_number(8.6)) # 🔴 10 # ---------- # 🔴 round a number DOWN to the nearest even number def round_down_to_nearest_even_number(num): return math.floor(num / 2) * 2 print(round_down_to_nearest_even_number(3.1)) # 🔴 2 print(round_down_to_nearest_even_number(8.6)) # 🔴 8 The code for this article is available on GitHubWe used the round() function to round a number to the nearest even integer.When passed a single argument, the round() function rounds to the nearest integer.main.pyCopied!print(round(22.4)) # 🔴 22 print(round(22.6)) # 🔴 23 This is a two-step process:Divide the number by 2 and round the result to the nearest integer.Multiply the result by 2 to get the nearest even integer.# Round a number Up to the nearest even number in PythonUse the math.ceil() method to round a number up to the nearest even number.main.pyCopied!import math def round_up_to_nearest_even_number(num): return math.ceil(num / 2) * 2 print(round_up_to_nearest_even_number(3.1)) # 🔴 4 print(round_up_to_nearest_even_number(8.6)) # 🔴 10 The math.ceil method returns the smallest integer greater than or equal to the provided number.main.pyCopied!import math print(math.ceil(14.01)) # 🔴 15 print(math.ceil(14.99)) # 🔴 15 If the passed-in number has a fractional part, the math.ceil method rounds the number up.This is a two-step process:Divide the number by 2 and round the result up to the nearest integer.Multiply the result by 2 to get the next even number.# Round a number Down to the nearest even number in PythonUse the math.floor() method to round a number down to the nearest even number.main.pyCopied!import math def round_down_to_nearest_even_number(num): return math.floor(num / 2) * 2 print(round_down_to_nearest_even_number(3.1)) # 🔴 2 print(round_down_to_nearest_even_number(8.6)) # 🔴 8 The code for this article is available on GitHubThe math.floor() method returns the largest integer less than or equal to the provided number.main.pyCopied!import math print(math.floor(9.99)) # 🔴 9 print(math.floor(9.01)) # 🔴 9 If the passed-in number has a fractional part, the math.floor method rounds the number down.This is a two-step process:Divide the number by 2 and round the result down to the nearest integer.Multiply the result by 2 to get the number rounded down to the nearest even integer.I've also written an article on how to round a float to N decimal places.# Additional ResourcesYou can learn more about the related topics by checking out the following tutorials: