Python Rounding

Summary: in this tutorial, you'll learn how to use the Python `round()` function to round a number.

Introduction to the Python round() function

Rounding means making a number simpler but keeping its value close to its original value. For example, 89 rounded to the nearest ten is 90 because 89 is closer to 90 than to 80.

To round a number in Python, you use the built-in `round()` function:

```
round(number [, ndigits])
```

The `round()` function rounds the `number` to the closest multiple of $10^{-ndigits}$.

In other words, the `round()` function returns the `number` rounded to `ndigits` precision after the decimal point.

If `ndigits` is omitted or `None`, the `round()` will return the nearest integer.

Python round() function examples

Let’s take some examples to understand the `round()` function better.

1) Python round() function examples

The following example uses the `round()` function without passing the `ndigits` :

```
round(1.25)
```

Output:

```
1
```

It returns an integer `1` .

However, if you pass `ndigits` as zero, the `round()` function returns a float (https://www.pythontutorial.net/advanced-python/python-float/) 1.0:

```
round(1.25, 0)
```

Output:

```
1.0
```

The following illustrates how the `round()` function works under the hood:

Since `ndigits` is zero, the `round()` function rounds the number `1.25` to the closet multiple of $10^{-(0)} = 1$.

2) Python round() function example with negative ndigits

The following example uses the `round()` function with negative `ndigits` :

```
round(15.5, -1)
```

Because `ndigits` is `-1` , the `round()` function rounds the number `15.5` to the closest multiple of `10` ($10^{-(-1)}$):

Since `15.5` is situated between `10` and `20` (multiple of `10`), it's closer to `10` . Therefore, the `round()` function returns `10` .

3) Python round() function example with ties

When you round a number situated in the middle of two numbers, Python cannot find the closest number.

For example, if you round the number `1.25` with n is `1` . There will be no closest number:

In this case, Python uses the IEEE 754 standard for rounding, called the banker's rounding.

In the banker's rounding, a number is rounded to the nearest value, with ties rounded to the nearest value with an **even least significant digit**.

> Generally, a least significant digit in a number is the right most digit.

The banker's rounding comes from the idea that statistically 50% sample of numbers are rounded up and 50% are rounded down.

For example:

```
round(1.25, 1)
```

It returns 1.2

Because the least significant digit of `1.2` is `2`, which is even:

Similarly, the rounding of `1.35` will return `1.4`:

```
round(1.35, 1)
```

Python uses banker's rounding but not rounding away from zero because it's less biased.

For example, if you average three numbers `1.5` , `2.5` , and `3.5` , the rounding away from zero returns `3` .0 while the banker's rounding returns `2.66` :

Number	**Banker's Rounding**	**Rounding away from zero**
1.5	2	2
2.5	2	3
3.5	4	4
Average	2.66666...	3.0

How to round away from zero

Python doesn't provide a direct way to round a number away from zero as you might expect. For example:

Number	**Rounding away from zero**
1.2	1
1.5	2

A common way to round a number away from zero is to use the following expression:

```
int(x + 0.5)
```

This expression works correctly for positive numbers. For example:

```
print(int(1.2 + 0.5))
print(int(1.5 + 0.5))
```

Output:

```
1
2
```

However, it doesn’t work for the negative numbers:

```
print(int(-1.2 + 0.5))
print(int(-1.5 + 0.5))
```

Output:

```
0
-1
```

For negative numbers, you should subtract 0.5 instead of adding it.

The following example works properly for the negative numbers:

```
print(int(-1.2 - 0.5))
print(int(-1.5 - 0.5))
```

The following defines a helper function that rounds up a number:

```python
def round_up(x):
    if x > 0:
        return int(x + 0.5)
    return int(x - 0.5)
```

The Python `math` module provides you with a function called `copysign()` :

```python
math.copysign(x, y)
```

The `copysign()` function returns the absolute value of `x` but the sign of `y` .

And you can use this `copysign()` function to develop a `round_up()` function without checking whether `x` is positive or negative:

```python
from math import copysign

def round_up(x):
    return int(x + copysign(0.5, x))
```

Summary

- Use the `round(number, ndigits)` function to round a `number` to the `ndigits` precision after the decimal point.