FLOATS

Rounding

The `round()` function

Python provides a built-in rounding function: `round(x, n=0)`

This will round the number `x` to the closest multiple of

you might think of this as rounding to a certain number of digits after the decimal point

which would work for positive `n`, but `n` can, in fact, also be negative!

In addition to truncate, floor, and ceiling, we can therefore also use rounding (with n = 0) to coerce a float to an integer number

If `n` is not specified, then it defaults to zero and `round(x)` will therefore return an `int`

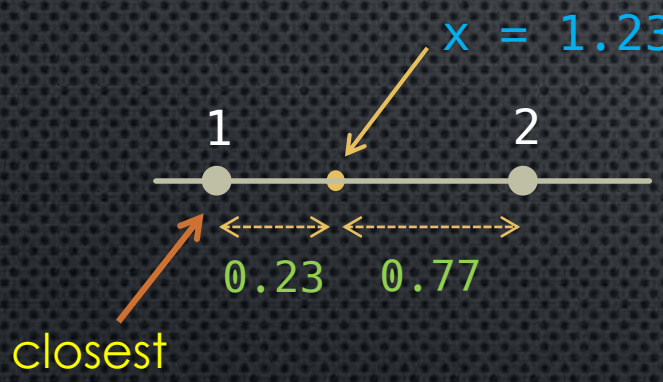
```
round(x) → int

round(x, n) → same type as x

round(x, 0) → same type as x
```

n = 0

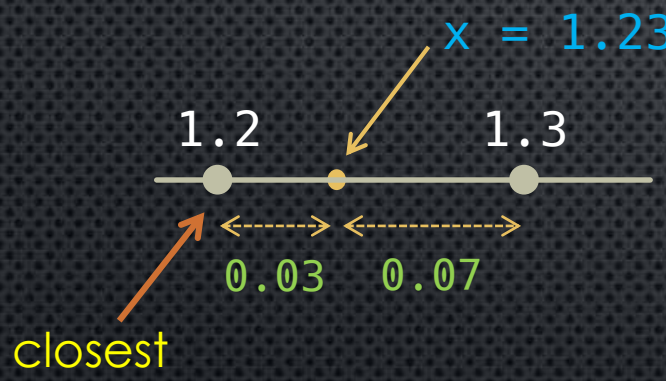
round to the closest multiple of



round(1.23) → 1

n > 0

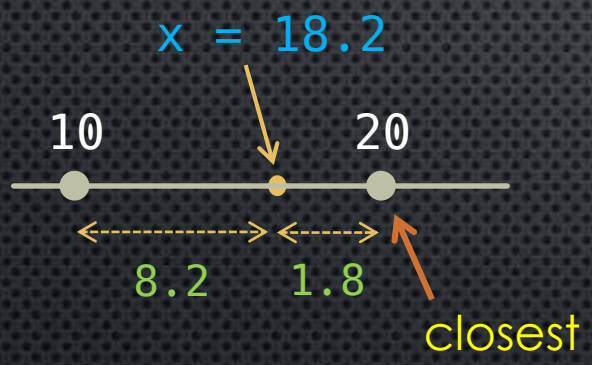
n = 1 round to the closest multiple of



round(1.23, 1) → 1.2

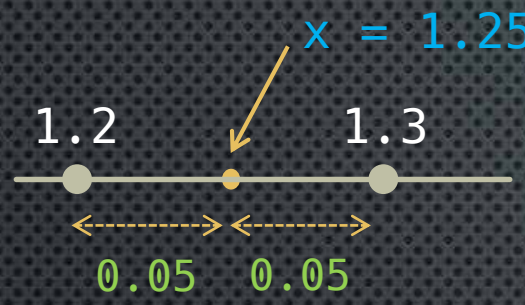
$n < 0$

`n = -1` round to the closest multiple of



`round(18.2, -1)` → `20`

Ties



`round(1.25, 1) = ???`

there is no closest value!!

We probably would expect `round(1.25, 1)` to be `1.3` rounding up / away from zero

Similarly, we would expect `round(-1.25, 1)` to result in `-1.3` rounding down / away from zero

This type of rounding is called rounding to nearest, with ties away from zero

But in fact:

`round(1.25, 1)` → `1.2` towards 0

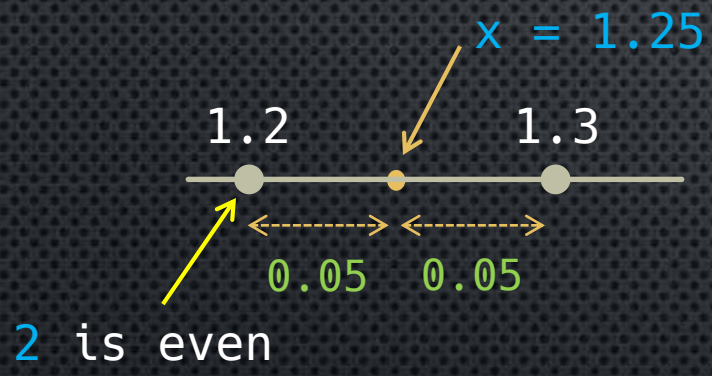
`round(1.35, 1)` → `1.4` away from 0

`round(-1.25, 1)` → `-1.2` towards 0

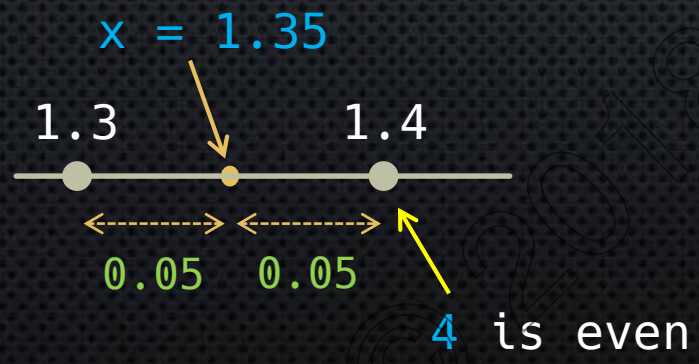
`round(-1.35, 1)` → `-1.4` away from 0

Banker's Rounding

IEEE 754 standard: rounds to the nearest value, with ties rounded to the nearest value with an even least significant digit

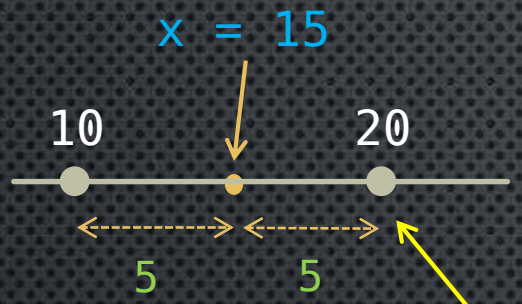


round(1.25, 1) → 1.2

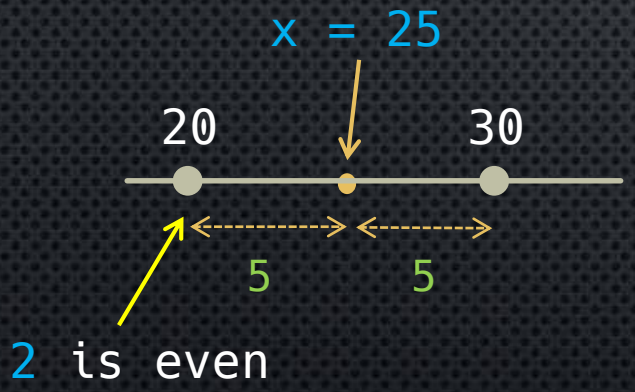


round(1.35, 1) → 1.4

`n = -1` round to the closest multiple of



`round(15, -1)` → `20`



`round(25, -1)` → `20`

Why Banker's Rounding?

Less biased rounding than ties away from zero

Consider averaging three numbers, and averaging the rounded value of each:

`0.5, 1.5, 2.5` → avg = 4.5 / 3 = 1.5

"standard" rounding: `1, 2, 3` → avg = 6 / 3 = 2

banker's rounding: `0, 2, 2` → avg = 4 / 3 = 1.3...

If you really insist on rounding away from zero…

One common (and partially incorrect) way to round to nearest unit that often comes up on the web is:

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

```
10.3 → int(10.3 + 0.5) = int(10.8) = 10
10.9 → int(10.9 + 0.5) = int(11.4) = 11
10.5 → int(10.5 + 0.5) = int(11.0) = 11
```

but, this does not work for negative numbers

```
-10.3 → int(-10.3 + 0.5) = int(-9.8) = -9
-10.9 → int(-10.9 + 0.5) = int(-10.4) = -10
-10.5 → int(-10.5 + 0.5) = int(-10.0) = -10
```

Technically, this is also an acceptable rounding method referred to as rounding towards + infinity

But this not rounding towards zero !!

If you really insist on rounding away from zero…

The correct way to do it:

```
sign(x) * int(abs(x)+0.5)
```

!! Not the same as the mathematical `sgn` (signum) function!

	10.4	10.5	10.6	-10.4	-10.5	-10.6
`sign(x)`	+	+	+	-	-	-
`abs(x)+0.5`	10.9	11.0	11.1	10.9	11.0	11.5
`int(abs(x)+0.5)`	10	11	11	10	11	11
`sign(x) * int(abs(x)+0.5)`	10	11	11	-10	-11	-11

```
= int(x + 0.5 * sign(x))
```

Python does not have a `sign` function !

We can however use the `math.copysign()` function to achieve our goal:

`copysign(x, y)` returns the magnitude (absolute value) of `x` but with the sign of `y`

```
sign(x) = copysign(1, x)
```

```
sign(x) * int(abs(x)+0.5)

def round_up(x):
    from math import fabs, copysign
    return copysign(1, x) * int(fabs(x) + 0.5)
```

A simpler way to code this:

```
int(x + 0.5 * sign(x))

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

Code