Numbers, bool and char

Rust basics · Numbers, bool and char

In PHP

PHP has one `int` and one `float`. `7 / 2` is `3.5`. `intdiv(7, 2)` is `3`. `PHP_INT_MAX + 1` becomes a float. The type grows.

These values are falsy: `0`, `0.0`, `""`, `"0"`, `[]`, `null`, and `false`. There is no `char`. `"é"` is a string, and `strlen` counts bytes.

numbers.php

```
 <?php

echo 7 / 2, "\n";
echo intdiv(7, 2), "\n";

$ready = "0";
if ($ready) {
    echo "truthy\n";
} else {
    echo "falsy\n";
}

echo strlen("é"), "\n";
```

That prints `3.5`, then `3`, then `falsy`, then `2`.

In Rust

An unsuffixed integer is `i32`. An unsuffixed float is `f64`. A suffix picks another width (`1_000_000u64`, `2.5_f32`). Underscores are only for reading.

`isize` and `usize` match the pointer width. A length or an index is a `usize`.

	8 bits	16	32	64	128

Signed	`i8`	`i16`	`i32`, the default	`i64`	`i128`
Unsigned	`u8`	`u16`	`u32`	`u64`	`u128`

A signed `iN` runs from `-2^(N-1)` through `2^(N-1) - 1`. An unsigned `uN` runs from `0` through `2^N - 1`. `i8` is `-128` through `127`. `u8` is `0` through `255`.

Integer `/` truncates. `7 / 2` is `3`.

In a debug build, an overflowing `+` panics. A release build wraps. `checked_add` returns `None` either way. Next to it, `wrapping_add` always wraps and `saturating_add` clamps at the type's min or max. The calculator uses `checked_add`, `checked_sub`, `checked_mul`, and `checked_div` so overflow and divide-by-zero both come back as `None`.

`as` between integers keeps the low bits. `300 as u8` is `44`. When the value might not fit, use `try_into` (or `TryFrom`). `as` from a float rounds toward zero, so `300.9 as i32` is `300`. A float that cannot fit saturates to the integer's min or max, and `NaN` becomes `0`. That cast does not report the loss.

Do not compare floats with `==`. `0.1 + 0.2` is not a clean `0.3`. Check an explicit tolerance instead: `(0.1 + 0.2 - 0.3).abs() < 1e-9`. `f64::EPSILON` is the gap between `1.0` and the next `f64`, so it is not a general-purpose tolerance.

`if` needs a `bool`. A number is not a condition.

`char` is four bytes and one Unicode scalar value. It is not a PHP string, and it is not a grapheme cluster.

src/main.rs

```
 use std::f64::consts::PI;

fn main() {
    let n = 42;
    let width = 1.5;
    let big = 1_000_000u64;
    let single = 2.5_f32;
    let index: usize = 0;
```

```
    println!("{n} {width} {big} {single} {index}");

    let quotient = 7 / 2;
    println!("{quotient}");

    let overflow = i32::MAX.checked_add(1);
    println!("{overflow:?}");

    let wide: i32 = 300;
    let truncated = wide as u8;
    println!("{truncated}");

    let fitted: Result<u8, _> = 200i32.try_into();
    let too_big: Result<u8, _> = wide.try_into();
    println!("{fitted:?} {too_big:?}");

    let toward_zero = 300.9_f64 as i32;
    println!("{toward_zero}");

    println!("{PI}");

    let ready = true;
    if ready {
        println!("ready");
    }

    let letter: char = 'é';
    let han: char = '?';
    let bytes = size_of::<char>();
    println!("{letter} {han} {bytes}");

    let close = (0.1_f64 + 0.2 - 0.3).abs() < 1e-9;
    println!("{close}");
}
```

`quotient` is `3`. `overflow` is `None`. `truncated` is `44`. `fitted` is `Ok(200)` and `too_big` is `Err`. `toward_zero` is `300`. `bytes` is `4`. `close` is `true`.

> **Watch for:** `as` keeps going when the value does not fit. `try_into` returns `Err` instead. A float compared with `==` will not do what the decimal looks like it should. `f64::EPSILON` is not a drop-in tolerance for every magnitude.

> **Idiomatic Rust:** `checked_add` when overflow matters, as in the calculator. `wrapping_add` and `saturating_add` when you want wrap or clamp instead of `None`. `usize` for an index. `std::f64::consts::PI` for pi. `(a - b).abs() < 1e-9` (or another tolerance you choose) for float equality checks.

Side by side

	PHP	Rust
`7 / 2`	`3.5` . `intdiv` is `3`	`3`
Overflow	The int becomes a float	Debug panics, release wraps, `checked_add` is `None`
Narrowing	The value may change type	`as` truncates. `try_into` when it might not fit
Float equality	Often `abs($a - $b) < 1e-9`	`(a - b).abs() < 1e-9` . Not `f64::EPSILON` as a general tolerance
`"0"` or `[]` in `if`	Falsy	`if` needs a `bool`
One letter	A string. `strlen("é")` is `2`	`char` , four bytes, one scalar

Try it

Open `src/main.rs` in the Playground. Confirm `7 / 2` prints `3` and `checked_add` prints `None` . Change `wide` from `300` to `40` and run it again. `as` and `try_into` then agree on `40` . Put `300` back: `as` prints `44` and `try_into` is `Err` . Confirm `close` prints `true` for the `1e-9` tolerance check.

Key takeaways

- Unsuffixed integers are `i32` and unsuffixed floats are `f64` . Width is the number of bits. `usize` is the index type.
- `7 / 2` is `3` . Overflow is `checked_add` when you need a `None` . Prefer `wrapping_add` or `saturating_add` when wrap or clamp is the rule.

- `as` truncates. `try_into` fails when the integer does not fit. Compare floats with an explicit tolerance such as `1e-9`, not `==`. `f64::EPSILON` is not a general-purpose tolerance.
- `bool` has no truthiness. `char` is one Unicode scalar, four bytes wide.

What's next: Functions and expressions