Variables

Rust basics · Variables

In PHP

A variable is mutable, and its name lives for the whole function. An `if` block does not hide it. After the brace, `$points` is still `2`.

`strlen("five")` is `4`.

`const` is a compile-time name. On PHP 8.3, assigning to it is a parse error: `syntax error, unexpected token "="`. A `readonly` property is different: the constructor sets it, and a later assignment throws `Cannot modify readonly property`.

score.php

```
 <?php

function score(): void
{
    $points = 1;
    if (true) {
        $points = 2;
    }
    echo $points, "\n";

    $count = "five";
    $count = strlen($count);
    echo $count, "\n";
}

score();

const MAX_COUNT = 100;
echo MAX_COUNT, "\n";
```

That prints `2`, then `4`, then `100`.

constants.php

```
 <?php

const MAX_COUNT = 100;
MAX_COUNT = 200;
```

PHP 8.3 rejects that file before it runs: `Parse error: syntax error, unexpected token "="`.

Config.php

```
 <?php

readonly class Config
{
    public function __construct(public int $maxCount)
    {
    }
}

$config = new Config(100);
echo $config->maxCount, "\n";
```

`$config->maxCount = 200` throws `Error: Cannot modify readonly property Config::$maxCount`.

In Rust

`let` binds a name. Without `mut`, a second assignment is a compile error. `let` again, in the same scope, creates a new binding. That is how a string becomes a number without a mutable variable: `parse` is the usual reason to shadow. The old binding is hidden for the rest of that scope.

A shadow inside a block ends at that block's `}`. The outer name is still there.

`const` is an item, not a `let` binding. It has a type, the name is `SCREAMING_CASE`, the value is computed at compile time, and it is never `mut`.

src/main.rs

```
 const MAX_POINTS: u32 = 100_000;

fn main() {
    let x = 5;
    println!("{x}");

    let mut counter = 0;
    counter += 1;
    println!("{counter}");

    let guess = "42";
    let guess: u32 = guess.parse().expect("guess should be a number");
```

```
    println!("{guess}");

    let count = "five";
    let count = count.len();
    println!("{count}");

    let name = 5;
    {
        let name = name + 1;
        println!("inner {name}");
    }
    println!("outer {name}");

    println!("{MAX_POINTS}");
}
```

`count` prints `4`. `inner` prints `6`. `outer` prints `5`.

> **Watch for:** A shadow inside a block ends at that block's `}`. The outer name comes back. In PHP the assignment would still be visible, because the name belongs to the function.

> **Idiomatic Rust:** Shadow with `parse` when a name should change type and stay immutable. Use `const` for a typed value the compiler can compute up front.

Side by side

	PHP	Rust
Second assignment	Updates the same variable	Needs `let mut`, or a new `let`
`"five"` length	`strlen` is `4`	`.len()` is `4`
Name inside `if` or `{ }`	Still visible after the block	A shadow lasts only until `}`
Constant	`const`. Assigning to it is a parse error	`const NAME: Type = value`. Never `mut`

	PHP	Rust
Immutable field	`readonly` property	Fields are immutable unless the whole value is `let mut`

A name you bind but never read is an unused-variable warning. Prefix it with `_` (`let _scratch = 1;`) to silence that, the same idea as intentionally unused parameters. You can also declare without a value and assign once: `let x; x = 5;`. That is still one immutable binding, not `mut`.

Try it

Open `src/main.rs` in the Playground. Change `"42"` to `"four"` and run it. `expect` panics because the parse failed. Put `"42"` back. Under `let x = 5;`, add `x = 6;` and read the compiler error. It is `E0384` (cannot assign twice to immutable variable). Remove that line. `outer` still prints `5`.

Key takeaways

- `let` is immutable. `let mut` is the assignment that changes one binding. A second assignment without `mut` is `E0384`.
- Shadowing is a new `let` in the same scope. `parse` is the usual reason. A shadow inside `{ }` ends at `}`.
- `const` is a typed item in `SCREAMING_CASE`. It is never `mut`. PHP reports `syntax error, unexpected token "="` if you assign to a `const`.
- `_name` silences unused-variable warnings. `let x; x = 5;` is deferred init of one immutable binding.

What's next: Numbers, bool and char