Control flow

Rust basics · Control flow

In PHP

`if` checks truthiness. These values are falsy: `0`, `0.0`, `""`, `"0"`, `[]`, `null`, and `false`. The same list is in the numbers lesson. A ternary is the expression form of `if`.

`range(5, 1)` counts down: `5 4 3 2 1`. A C-style `for` keeps the index in `$i`. `break 2` leaves two loops. `do { } while` runs the body, then tests.

flow.php

```php
 <?php

$n = 7;
$label = $n > 0 ? "positive" : "non-positive";
echo $label, "\n";

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

foreach (range(5, 1) as $value) {
    echo $value, " ";
}
echo "\n";

$nums = [10, 20, 30];
for ($i = 0; $i < count($nums); $i++) {
    echo $i, ": ", $nums[$i], "\n";
}

for ($x = 0; $x < 3; $x++) {
    for ($y = 0; $y < 3; $y++) {
        if ($x === 1 && $y === 1) {
            break 2;
        }
        echo "$x,$y\n";
    }
}
```

In Rust

`if` is an expression. Both branches have the same type, and the condition is a `bool`. `if 1` does not compile. `if "0"` does not compile.

`loop` runs until `break`. `break` can carry the value of the loop. `while` tests a `bool` before each pass. `for` walks a range or a shared collection (`for item in &nums`).

`1..4` is `1, 2, 3` . `1..=5` includes `5` . `(1..=5).rev()` counts down. `5..1` is an empty range: **rustc accepts it**, and the loop body never runs. Clippy is what rejects it, via `clippy::reversed_empty_ranges` , which is deny by default when you run `cargo clippy` (or a setup that enables Clippy):

```
 error: this range is empty so it will yield no values
  --> src/main.rs:2:18
   |
2 |     for value in 5..1 {
   |                  ^^^^
   |
   = note: `#[deny(clippy::reversed_empty_ranges)]` on by default
help: consider using the following if you are attempting to iterate over this
range in reverse
   |
2 -     for value in 5..1 {
2 +     for value in (1..5).rev() {
```

`(1..5).rev()` is `4, 3, 2, 1` . Include both ends, as PHP `range(5, 1)` does, with `(1..=5).rev()` .

`.enumerate()` gives the index. A label such as `'outer:` lets `break 'outer` leave the outer loop. PHP's `break 2` is that idea. `continue` skips the rest of this pass. `continue 'outer` starts the next pass of the labelled loop. Rust has no `do while` . A `loop` with `break` at the end of the body is that shape.

`while let` is the idiomatic loop for "keep going while this pattern matches". PHP developers often miss it. Popping a stack until it is empty is:

```
 let mut stack = vec![1, 2, 3];
while let Some(top) = stack.pop() {
    println!("{top}");
}
```

src/main.rs

```
fn main() {
    let n = 7;
    let label = if n > 0 { "positive" } else { "non-positive" };
    println!("{label}");

    let mut ticks = 0;
    let found = loop {
        ticks += 1;
        if ticks == 4 {
            break ticks;
        }
    };
    println!("{found}");

    let mut i = 0;
    while i < 3 {
        i += 1;
    }
    println!("{i}");

    for value in 1..4 {
        println!("{value}");
    }
    for value in (1..=5).rev() {
        println!("rev {value}");
    }

    let nums = [10, 20, 30];
    for item in &nums {
        println!("{item}");
    }
    for (index, item) in nums.iter().enumerate() {
        println!("{index}: {item}");
    }

    'outer: for x in 0..3 {
        for y in 0..3 {
            if x == 1 && y == 1 {
                break 'outer;
            }
            println!("{x},{y}");
        }
    }

    let mut step = 0;
    loop {
        step += 1;
        println!("step {step}");
        if step == 2 {
            break;
        }
    }

    let mut stack = vec![1, 2, 3];
```

```
    while let Some(top) = stack.pop() {
        println!("pop {top}");
    }
}
```

`label` is `positive`. `found` is `4`. `1..4` prints `1`, `2`, `3`. The countdown prints `rev 5` through `rev 1`. The labelled loops print `0,0`, `0,1`, `0,2`, and `1,0`, then stop. `step` prints `1` and `2`. The stack prints `pop 3`, `pop 2`, `pop 1`.

> **Watch for:** `5..1` compiles with `rustc` and yields nothing. Clippy denies it. Count down with `(1..=5).rev()`. An `if` condition is a `bool`, including the falsy PHP values `"0"` and `[]`.

> **Idiomatic Rust:** `for item in &nums`, and `.enumerate()` when you need the index. `while let` for pattern loops. `break 'outer` / `continue 'outer` on a labelled loop. `break` with a value when the loop's result is a number.

Side by side

	PHP	Rust
`if` as a value	Ternary	`if` / `else`, both branches the same type
Condition	Falsy list, including `"0"` and `[]`	A `bool`
Count down	`range(5, 1)`	`(1..=5).rev()`. `5..1` is empty (clippy denies it)
Index	`for ($i = 0; ...)`	`.enumerate()`
Leave two loops	`break 2`	`break 'outer`
Pattern loop	`while (($x = array_pop(...)) !== null)`	`while let Some(x) = ...`

	PHP	Rust
`do while`	Body, then the test	`loop`, with `break` at the end

Try it

Open `src/main.rs` in the Playground. `1..4` prints `1`, `2`, and `3`. `(1..=5).rev()` prints `5` down to `1`. Change `break 'outer` to `break` and run it again. The inner loop stops at `1,1`, and the outer loop continues with `2,0`.

Key takeaways

- `if` is an expression. The condition is a `bool`, and both branches have one type.
- `loop` can `break` a value. `while` tests first. `while let` keeps going while the pattern matches. `for` walks `..`, `..=`, or `&collection`.
- `5..1` compiles with `rustc` and yields nothing. Clippy denies `reversed_empty_ranges`. `(1..=5).rev()` counts down the way `range(5, 1)` does. `.enumerate()` replaces `$i`.
- `'outer:` with `break 'outer` or `continue 'outer` targets the labelled loop. There is no `do while`. Put `break` at the end of a `loop`.

What's next: Tuples and arrays