Tuples and arrays

Rust basics · Tuples and arrays

In PHP

A list can mix types. `[$left, $right] = pair()` takes the two values a function returns. `count` is the length, and the length can change later by appending. Nothing in the type says "this list has 5 elements".

pairs.php

```
 <?php

function pair(int $left, int $right): array
{
    return [$left + $right, $left * $right];
}

[$sum, $product] = pair(3, 4);
echo "$sum $product\n";

$point = [10, 2.5, "origin"];
echo $point[0], " ", $point[1], " ", $point[2], "\n";

$zeros = array_fill(0, 5, 0);
echo count($zeros), "\n";
```

In Rust

A tuple packs mixed types. `.0`, `.1`, and `.2` read the fields. Destructuring names them. A function returns two values as one tuple, the way `[$sum, $product] = pair()` does in PHP.

An array `[T; N]` is one type of element, and the length is part of the type. `[0; 5]` is five zeros. `.len()` is the length. `for n in &numbers` walks the elements.

`numbers[10]` does not compile. `let index = 10; numbers[index]` does not compile either: rustc can see the `10`.

```
 error: this operation will panic at runtime
  --> src/main.rs:3:20
   |
3 |     println!("{}", numbers[10]);
   |                    ^^^^^^^^^^^ index out of bounds: the length is 5 but
the index is 10
   |
   = note: `#[deny(unconditional_panic)]` on by default
```

An index the compiler cannot fold still compiles, then panics when it runs.

`std::hint::black_box(10)` is one way to hide a value from rustc's const analysis:

```
 thread 'main' panicked at src/main.rs:4:20:
index out of bounds: the len is 5 but the index is 10
```

`.get(10)` returns `Option`. Here it is `None`.

A slice is a view of contiguous elements: `&[T]`. `&numbers[1..3]` is a two-element borrow of the array (indexes 1 and 2). The next lessons pass `&[Option<i32>]` the same way. Chapter 3 goes deeper on borrowing. When the length has to change, use `Vec`. That is the next lesson.

src/main.rs

```
 fn pair(left: i32, right: i32) -> (i32, i32) {
    (left + right, left * right)
}

fn main() {
    let point = (10, 2.5, "origin");
    let x0 = point.0;
    let y0 = point.1;
    let label0 = point.2;
    println!("{x0} {y0} {label0}");

    let (x, y, label) = point;
    println!("{x} {y} {label}");

    let (sum, product) = pair(3, 4);
    println!("{sum} {product}");

    let zeros = [0; 5];
    let len = zeros.len();
    println!("{len} {zeros:?}");

    let numbers = [10, 20, 30, 40, 50];
    for n in &numbers {
        println!("{n}");
```

```
    }

    let index = 1;
    let chosen = numbers[index];
    println!("{chosen}");
    let missing = numbers.get(10);
    println!("{missing:?}");

    let mid: &[i32] = &numbers[1..3];
    println!("{mid:?}");
}
```

`pair(3, 4)` prints `7` and `12`. `zeros` prints `5 [0, 0, 0, 0, 0]`. `chosen` is `20`. `missing` is `None`. `mid` is `[20, 30]`.

To see the runtime panic, replace `let index = 1;` with `let index = std::hint::black_box(10);` and run it. The index is 10, the length is 5, and the program panics.

> **Watch for:** `numbers[10]` is a compile error on a `[T; N]`. A `let` that rustc can see through is the same error. Hide the value with `black_box` (or parse it from args) when you need a real runtime panic. `.get` is the call that returns `None` instead of panicking.

> **Idiomatic Rust:** Return `(sum, product)` when a function has two results. `for n in &numbers` to walk an array. `&numbers[1..3]` for a slice. `.get(index)` when the index comes from outside the program.

Side by side

	PHP	Rust
Mixed values	One array	A tuple, read with `.0` or destructuring
Two return values	`return [a, b]` and `[$a, $b] = f()`	`-> (i32, i32)`

	PHP	Rust
Fixed list	`array_fill`, length can still change	`[T; N]`. `[0; 5]`
Out of range	A warning, then `null`	Constant index: compile error. Opaque runtime index: panic. `.get` is `Option`
Slice of a list	`array_slice`	`&numbers[1..3]` is `&[T]`
Growing list	`$a[] = 1`	`Vec`, next lesson

Try it

Open `src/main.rs` in the Playground. `chosen` prints `20` and `missing` prints `None`. Change `index` to `std::hint::black_box(10)` and run it. The program panics: the length is 5 and the index is 10. Put `1` back.

Key takeaways

- A tuple holds mixed types. `.0` reads a field, and destructuring names the fields. Two return values are one tuple.
- `[T; N]` fixes the length in the type. `[0; 5]` repeats a value. `for n in &numbers` walks it. `&numbers[1..3]` is a slice (`&[T]`).
- `numbers[10]` does not compile. An index rustc cannot precompute panics. `.get` returns `Option`.
- A list whose length changes is a `Vec`.

What's next: Vec and HashMap