Pattern Matching

Pattern matching in Epos replaces traditional if-else statements and provides powerful control flow based on value structure.

Basic Match Expressions

The fundamental syntax uses `match ... then ... end`:

```
number: int = 42
message: string = match number then
    0 => "zero"
    1 => "one"
    42 => "the answer"
    _ => "something else"
end
```

Multiple Value Matching

Match against multiple possible values:

```
day: int = 3
day-type: string = match day then
    1, 2, 3, 4, 5 => "weekday"
    6, 7 => "weekend"
    _ => "invalid day"
end
```

Boolean Matching

Pattern match on boolean conditions:

```
age: int = 20
status: string = match age >= 18 then
    true => "adult"
    false => "minor"
end
```

List Pattern Matching

Match on list structure and contents:

```
numbers: list(int) = {1, 2, 3}
empty: list(int) = {}

result: string = match numbers then
    empty => "empty list"
    {1} => "just one"
    {1, 2} => "two elements starting with 1"
    _ => "something else"
end

# Pattern match with list length
fn describe-list(items: list(t)): string
    match len(items) then
        0 => "empty"
        1 => "single item"
        _ => "multiple items"
    end
end
```

Using Match in Control Flow

Since Epos doesn't have traditional loops, use pattern matching with recursion:

```
fn countdown(n: int)
    print(n)
    match n <= 0 then
        true => print("Done!")
        false => countdown(n - 1)
    end
end

fn process-list(items: list(string), index: int = 0)
    match index < len(items) then
        true => {
            print(elem(items, index))
            process-list(items, index + 1)
        }
        false => print("All done")
    end
end
```

Record Pattern Matching

Match on record structures and fields:

```
record Status
    code: int
    message: string
end

fn handle-response(status: Status): string
    match status.code then
        200 => "Success: #{status.message}"
        404 => "Not Found"
        500 => "Server Error: #{status.message}"
        _ => "Unknown status: #{status.code}"
    end
end

response: Status = @{
    code => 200,
    message => "OK"
}
result := handle-response(response)
```

Nested Pattern Matching

Pattern matching can be nested for complex logic:

```
record User
    name: string
    age: int
    is-admin: bool
end

fn check-access(user: User): string
    match user.is-admin then
        true => "Full access granted"
        false => match user.age >= 18 then
```

```
            true => "Limited access granted"
            false => "Access denied"
        end
    end
end
```

Pattern Matching with Functions

Use pattern matching to create different behaviors:

```
fn fibonacci(n: int): int
    match n then
        0 => 0
        1 => 1
        _ => fibonacci(n - 1) + fibonacci(n - 2)
    end
end

fn factorial(n: int): int
    match n <= 1 then
        true => 1
        false => n * factorial(n - 1)
    end
end
```

Expression-Based Control

Since match is an expression, it can be used anywhere a value is expected:

```
# In variable assignment
tax-rate: float = match income then
    10000 => 0.0
    50000 => 0.15
    _ => 0.25
end

# In function arguments
print(match weather then
    "sunny" => "Wear sunglasses!"
    "rainy" => "Bring an umbrella!"
    _ => "Have a nice day!"
end)

# As return values
fn get-discount(customer-type: string): float
    match customer-type then
        "premium" => 0.20
        "regular" => 0.10
        "new" => 0.05
        _ => 0.0
    end
end
```

Pattern matching is the primary control flow mechanism in Epos, replacing traditional if-else statements and loops with more expressive and functional constructs.

Next, learn about advanced features in Epos.