# Módulo Time

O módulo **time** fornece funções para manipulação de tempo e datas. Este módulo é útil para scripts que precisam medir intervalos, aguardar por períodos específicos, parsear datas e implementar timeouts em operações assíncronas.

**Características principais:**

- Timestamps Unix em segundos
- Parse de datas no formato RFC 3339
- Sleep assíncrono não-bloqueante
- Timeouts para operações assíncronas

# Funções Disponíveis

## 1. `time.now()`

Retorna o timestamp Unix atual em segundos.

### Parâmetros:

- Nenhum

### Retorno:

- **número**: Timestamp Unix atual (segundos desde 1 de Janeiro de 1970, 00:00:00 UTC)

### Comportamento:

- Retorna o tempo atual em UTC
- Precisão de segundos (não milissegundos)
- Mesmo valor retornado por `os.time()` em Lua padrão
- Disponível como função global e no módulo `time`

### Exemplo de Uso:

```
-- Usando o módulo time
```

```
local timestamp_modulo = time.now()
print("Timestamp módulo:", timestamp_modulo)

-- Usando a função global (mesma função)
local timestamp_global = now()
print("Timestamp global:", timestamp_global)

-- Ambos retornam o mesmo valor
print("São iguais?", timestamp_global == timestamp_modulo)  -- true

-- Calcular duração de operação
local inicio = time.now()
-- ... operação demorada ...
local fim = time.now()
local duracao = fim - inicio
print("Operação levou", duracao, "segundos")

-- Registrar eventos com timestamp
local evento = {
    tipo = "login",
    usuario = "admin",
    timestamp = now(),
    origem = system.hostname()
}

-- Agendar execução futura
local proxima_execucao = time.now() + 3600  -- 1 hora a partir de agora
print("Próxima execução em:", os.date("%H:%M:%S", proxima_execucao))
```

# 2. `time.sleep(segundos)`

Aguarda assincronamente por um número específico de segundos.

## Parâmetros:

- **segundos** (número): Número de segundos para aguardar

## Retorno:

- **nil**: A função não retorna valor (após a espera)

## Comportamento:

- Precisão dependente do scheduler do sistema

## Exemplo de Uso:

```
-- Aguardar 5 segundos
await(time.sleep(5))
print("5 segundos se passaram")

-- Implementar polling com intervalo
local function poll_com_intervalo(url, intervalo_segundos, max_tentativas)
    for tentativa = 1, max_tentativas do
        log.info("Tentativa", tentativa, "de", max_tentativas)

        -- Fazer requisição
        local resposta, status = http.get(url)

        if status == 200 then
            log.info("Sucesso na tentativa", tentativa)
            return resposta
        end

        -- Aguardar antes da próxima tentativa
        if tentativa < max_tentativas then
            log.info("Aguardando", intervalo_segundos, "segundos...")
            time.sleep(intervalo_segundos)
        end
    end

    log.error("Todas as tentativas falharam")
    return nil
end

-- Uso
local dados = poll_com_intervalo("https://api.exemplo.com/status", 10, 6)

-- Controle de execução com backoff exponencial
local function executar_com_backoff(funcao, max_tentativas)
    local tentativa = 1
```

```
    while tentativa <= max_tentativas do
        local ok, resultado = pcall(funcao)

        if ok then
            return resultado
        end

        -- Backoff exponencial: 2^(tentativa-1) segundos
        local wait_time = math.pow(2, tentativa - 1)
        log.warn("Tentativa", tentativa, "falhou. Aguardando", wait_time, "segundos")

        if tentativa < max_tentativas then
            time.sleep(wait_time)
        end

        tentativa = tentativa + 1
    end

    error("Todas as tentativas falharam")
end
```

# 3. `time.parse(string_data)`

Parseia uma string de data/hora no formato RFC 3339 e retorna um objeto de data.

## Parâmetros:

- **string_data** (string): Data/hora no formato RFC 3339 (ex: "2024-01-15T10:30:00Z")

## Retorno:

- **objeto LuaDateTime**: Objeto com método `seconds()` que retorna timestamp Unix

## Formato RFC 3339:

- `YYYY-MM-DDTHH:MM:SSZ` (UTC)
- `YYYY-MM-DDTHH:MM:SS+HH:MM` (com offset de fuso horário)
- Exemplos: "2024-01-15T10:30:00Z", "2024-01-15T07:30:00-03:00"

## Exemplo de Uso:

```lua
-- Parsear data UTC
local data_utc = time.parse("2024-01-15T10:30:00Z")
local timestamp_utc = data_utc:seconds()
print("Timestamp UTC:", timestamp_utc)  -- 1705314600

-- Parsear data com offset de fuso
local data_local = time.parse("2024-01-15T07:30:00-03:00")
local timestamp_local = data_local:seconds()
print("Timestamp local:", timestamp_local)  -- 1705314600 (mesmo momento)

-- Converter para formato legível
local function formatar_data_rfc3339(timestamp)
    return os.date("%Y-%m-%dT%H:%M:%SZ", timestamp)
end

-- Calcular diferença entre datas
local data1 = time.parse("2024-01-15T10:30:00Z")
local data2 = time.parse("2024-01-15T11:45:00Z")
local diferenca = data2:seconds() - data1:seconds()
print("Diferença:", diferenca, "segundos (", diferenca/60, "minutos)")

-- Validar e parsear data de entrada
local function parsear_data_segura(data_string)
    local ok, data = pcall(time.parse, data_string)
    if ok then
        return data:seconds()
    else
        log.error("Data inválida:", data_string, "-", data)
        return nil
    end
end

-- Uso
local timestamp = parsear_data_segura("2024-01-15T10:30:00Z")
if timestamp then
    print("Data válida:", os.date("%d/%m/%Y %H:%M:%S", timestamp))
end
```

Revision #1
Created 3 February 2026 16:47:18 by Marc
Updated 3 February 2026 16:47:31 by Marc