**Usando o Googlesheet como BASEDADOS e Appscript para leituras rápidas, ficam aqui os códigos e explicações dos mesmos, do projecto appscript:**

Conteúdos AppScript

[*Funções* 2](#_Toc208230660)

[*enviarParaFirebase* 2](#_Toc208230661)

[código: 2](#_Toc208230662)

[explicação: 12](#_Toc208230663)

[*importarFirebaseParaSheet* 13](#_Toc208230664)

[código: 13](#_Toc208230665)

[explicação: 15](#_Toc208230666)

[*eliminarIncompletos* 16](#_Toc208230667)

[código: 16](#_Toc208230668)

[explicação: 17](#_Toc208230669)

[*EliminarBDFirebase* 17](#_Toc208230670)

[código: 17](#_Toc208230671)

[explicação: 18](#_Toc208230672)

[*Observação.gs* 18](#_Toc208230673)

[código: 18](#_Toc208230674)

[explicação: 19](#_Toc208230675)

# *Funções*

# *enviarParaFirebase*

## código:

/\*\*

 \* Sincroniza a folha "Firebase" com a coleção Firestore "acoesDividendos".

 \* - Em cada execução, PATCH de \*\*todos os campos\*\* (nome, ticker, setor, mercado, valorStock, periodicidade, mes, dividendo, taxas, origem).

 \* - Para \*\*não-ETFs\*\*, tenta enriquecer `dividendo` via Alpha Vantage em batches (respeita rate limit).

 \* - Para ETFs, usa sempre o valor da sheet (não chama Alpha).

 \* - Se o doc não existir, faz CREATE completo.

 \*/

// ======= Config =======

const USE\_ALPHA        = true;               // ativa/desativa a consulta à Alpha Vantage (para não-ETFs)

const BATCH\_SIZE       = 25;                 // nº de tickers enriquecidos por execução (Alpha)

const ALPHA\_SLEEP\_MS   = 12000;              // 12s entre chamadas -> ~5/min (plano gratuito)

const PROP\_KEY\_OFFSET  = 'ALPHA\_OFFSET';     // offset rotativo para o batching

const ALPHA\_PROP\_KEY   = 'ALPHA\_VANTAGE\_KEY';// nome da chave nas Script Properties

function enviarParaFirebase() {

  var ss = SpreadsheetApp.getActiveSpreadsheet();

  var sheet = ss.getSheetByName("Firebase");

  if (!sheet) {

    Logger.log("❌ Folha 'Firebase' não foi encontrada.");

    return;

  }

  // Lê dados: A..K (11 colunas), a partir da linha 2

  var lastRow = sheet.getLastRow();

  if (lastRow < 2) {

    Logger.log("⚠️ Sem dados para processar.");

    return;

  }

  var data = sheet.getRange(2, 1, lastRow - 1, 11).getValues();

  // Autorização (usar token do Apps Script)

  var headers = { "Authorization": "Bearer " + ScriptApp.getOAuthToken() };

  // 1) Lê documentos existentes (map por TICKER UPPERCASE)

  var firestoreUrl = "https://firestore.googleapis.com/v1/projects/appfinance-812b2/databases/(default)/documents/acoesDividendos?pageSize=1000";

  var response = UrlFetchApp.fetch(firestoreUrl, { method: "get", headers: headers, muteHttpExceptions: true });

  if (response.getResponseCode() !== 200) {

    Logger.log("❌ Erro ao listar Firestore: " + response.getContentText());

    return;

  }

  var docs = JSON.parse(response.getContentText());

  var firestoreDocs = docs.documents || [];

  var firestoreMap = {}; // TICKER -> { docName, fields }

  firestoreDocs.forEach(function(doc) {

    var fields = doc.fields || {};

    var tk = fields.ticker && fields.ticker.stringValue ? String(fields.ticker.stringValue).toUpperCase() : "";

    if (tk) {

      firestoreMap[tk] = { docName: doc.name, fields: fields };

    }

  });

  // 2) Preparar janela/batching para Alpha (apenas não-ETFs)

  var props = PropertiesService.getScriptProperties();

  var alphaKey = props.getProperty(ALPHA\_PROP\_KEY);

  var useAlphaThisRun = USE\_ALPHA && !!alphaKey;

  var elegiveis = []; // índices da folha que são não-ETF

  for (var r0 = 0; r0 < data.length; r0++) {

    var nome0   = String(data[r0][0] || "");

    var setor0  = String(data[r0][2] || "");

    var tk0     = String(data[r0][1] || "").trim().toUpperCase();

    if (!tk0) continue;

    if (!isETF(nome0, setor0)) elegiveis.push(r0);

  }

  var offset = parseInt(props.getProperty(PROP\_KEY\_OFFSET) || '0', 10);

  if (!Number.isFinite(offset) || offset < 0 || offset >= Math.max(elegiveis.length,1)) offset = 0;

  var windowLen = Math.min(BATCH\_SIZE, elegiveis.length);

  var alphaIdxSet = new Set();

  for (var iwin = 0; iwin < windowLen; iwin++) {

    var idx = elegiveis[(offset + iwin) % Math.max(elegiveis.length,1)];

    alphaIdxSet.add(idx);

  }

  // 3) Loop principal: para cada linha, compõe payload completo e faz CREATE/PATCH

  for (var r = 0; r < data.length; r++) {

    var row = data[r];

    var nome          = row[0];

    var tickerIn      = row[1];

    var setor         = row[2];

    var mercado       = row[3];

    var valorStock    = row[4];

    var periodicidade = row[5];

    var mes           = row[6];

    var dividendo     = row[7]; // pode ser enriquecido

    var taxa1semana   = row[8];

    var taxa1mes      = row[9];

    var taxa1ano      = row[10];

    var ticker = String(tickerIn || "").trim().toUpperCase();

    if (!ticker || !String(nome).trim()) {

      Logger.log("⚠️ Linha ignorada (nome/ticker vazio). Linha " + (r + 2));

      continue;

    }

    // 3.1) Enriquecer dividendo via Alpha (apenas não-ETF + dentro da janela desta execução)

    var finalDividendo = toNumber(dividendo); // default: valor da sheet

    var eETF = isETF(nome, setor);

    if (!eETF && useAlphaThisRun && alphaIdxSet.has(r)) {

      try {

        var alpha = getDividendDataAlphaSafe(ticker, alphaKey);

        if (alpha && typeof alpha.dividendo === "number") {

          finalDividendo = toNumber(alpha.dividendo);

          // \*\*Se quiseres também alinhar mes/periodicidade com a Alpha, descomenta:\*\*

          // mes           = alpha.mes || mes;

          // periodicidade = alpha.periodicidade || periodicidade;

        }

      } catch (e) {

        Logger.log("⚠️ Alpha falhou para " + ticker + ": " + e);

      }

      // respeitar rate limit (5/min)

      Utilities.sleep(ALPHA\_SLEEP\_MS);

    }

    // 3.2) Payload completo (normalizado)

    var payloadFields = buildFullPayload(

      nome, ticker, setor, mercado,

      valorStock,

      periodicidade, mes,

      finalDividendo,

      taxa1semana, taxa1mes, taxa1ano

    );

    // 3.3) CREATE se não existir; caso exista, PATCH completo (todos os campos)

    var existing = firestoreMap[ticker];

    try {

      if (!existing) {

        var createUrl = "https://firestore.googleapis.com/v1/projects/appfinance-812b2/databases/(default)/documents/acoesDividendos?documentId=" + encodeURIComponent(ticker);

        var createResp = UrlFetchApp.fetch(createUrl, {

          method: "post",

          contentType: "application/json",

          payload: JSON.stringify({ fields: payloadFields }),

          headers: headers,

          muteHttpExceptions: true

        });

        if (createResp.getResponseCode() >= 200 && createResp.getResponseCode() < 300) {

          Logger.log("📄 Criado: " + ticker);

        } else {

          Logger.log("❌ Falha ao criar " + ticker + ": " + createResp.getContentText());

        }

      } else {

        var updateUrl = "https://firestore.googleapis.com/v1/" + existing.docName;

        var updateResp = UrlFetchApp.fetch(updateUrl, {

          method: "patch",

          contentType: "application/json",

          payload: JSON.stringify({ fields: payloadFields }),

          headers: headers,

          muteHttpExceptions: true

        });

        if (updateResp.getResponseCode() >= 200 && updateResp.getResponseCode() < 300) {

          Logger.log("🔄 Atualizado: " + ticker);

        } else {

          Logger.log("❌ Falha ao atualizar " + ticker + ": " + updateResp.getContentText());

        }

      }

    } catch (e) {

      Logger.log("❌ Erro em " + ticker + ": " + e);

    }

  }

  // 4) Avança o offset da janela para a próxima execução (apenas se houver elegíveis)

  if (elegiveis.length > 0) {

    var nextOffset = (offset + windowLen) % elegiveis.length;

    props.setProperty(PROP\_KEY\_OFFSET, String(nextOffset));

    Logger.log("✅ Concluído. Próximo offset Alpha: " + nextOffset + " / " + elegiveis.length);

  } else {

    Logger.log("✅ Concluído. (Sem não-ETFs elegíveis para Alpha)");

  }

}

/\*\* Heurística simples para detetar ETFs via nome/setor \*/

function isETF(nome, setor) {

  var n = String(nome || "").toLowerCase();

  var s = String(setor || "").toLowerCase();

  if (s.startsWith("etf")) return true;

  if (n.includes("etf")) return true;

  if (n.includes("ucits")) return true;

  if (n.includes("index fund")) return true;

  if (n.includes("trust")) return true; // p.ex. QQQ Trust

  return false;

}

/\*\* Constrói payload completo (normalizado) para Firestore \*/

function buildFullPayload(nome, ticker, setor, mercado, valorStock, periodicidade, mes, dividendo, taxa1semana, taxa1mes, taxa1ano) {

  return {

    nome:                         { stringValue: String(nome) },

    ticker:                       { stringValue: String(ticker).toUpperCase() },

    setor:                        { stringValue: String(setor || "") },

    mercado:                      { stringValue: String(mercado || "") },

    valorStock:                   { doubleValue: toNumber(valorStock) },

    periodicidade:                { stringValue: normalizaPeriodicidade(periodicidade) },

    mes:                          { stringValue: normalizaMes(mes) },

    dividendo:                    { doubleValue: toNumber(dividendo) },

    taxaCrescimento\_1semana:      { doubleValue: toNumber(taxa1semana) },

    taxaCrescimento\_1mes:         { doubleValue: toNumber(taxa1mes) },

    taxaCrescimento\_1ano:         { doubleValue: toNumber(taxa1ano) },

    origem:                       { stringValue: "sheets" }

  };

}

/\*\*

 \* Alpha Vantage (robusto): retry/backoff leve.

 \* Devolve { dividendo:Number, mes:StringPT, periodicidade:StringPT } ou null.

 \*/

function getDividendDataAlphaSafe(ticker, apiKey) {

  var MAX\_TRIES = 3;

  var lastErr = null;

  for (var i = 0; i < MAX\_TRIES; i++) {

    try {

      return getDividendDataAlpha(ticker, apiKey);

    } catch (e) {

      lastErr = e;

      Utilities.sleep(1500 \* (i + 1)); // backoff 1.5s, 3s

    }

  }

  if (lastErr) throw lastErr;

  return null;

}

/\*\*

 \* Busca dados de dividendos na Alpha Vantage e tenta inferir:

 \* - dividendo (valor do último pagamento)

 \* - mes (PT) do último pagamento

 \* - periodicidade (Mensal/Trimestral/Semestral/Anual) por heurística

 \* Se não conseguir, devolve null.

 \*/

function getDividendDataAlpha(ticker, apiKey) {

  var key = apiKey || PropertiesService.getScriptProperties().getProperty(ALPHA\_PROP\_KEY);

  if (!key) return null;

  var url = "https://www.alphavantage.co/query?function=TIME\_SERIES\_MONTHLY\_ADJUSTED&symbol=" + encodeURIComponent(ticker) + "&apikey=" + encodeURIComponent(key);

  var resp = UrlFetchApp.fetch(url, { method: "get", muteHttpExceptions: true });

  var code = resp.getResponseCode();

  if (code !== 200) throw new Error("Alpha HTTP " + code + " para " + ticker);

  var json = JSON.parse(resp.getContentText());

  var series = json["Monthly Adjusted Time Series"];

  if (!series) return null;

  // Extrai últimos meses com dividendos > 0

  var months = Object.keys(series).sort().reverse(); // datas "YYYY-MM"

  var divMonths = [];

  for (var i = 0; i < months.length; i++) {

    var mKey = months[i];

    var dAmt = Number(series[mKey]["7. dividend amount"] || 0);

    if (dAmt > 0) divMonths.push({ date: mKey, amount: dAmt });

    if (divMonths.length >= 6) break; // chega para inferir periodicidade

  }

  if (divMonths.length === 0) return null;

  // Último dividendo

  var last = divMonths[0];

  var lastMonthNum = Number(last.date.split("-")[1]); // "YYYY-MM"

  var mesNome = mesNomePt(lastMonthNum);

  // Heurística de periodicidade

  var difs = [];

  for (var j = 1; j < divMonths.length; j++) {

    var a = divMonths[j - 1].date.split("-");

    var b = divMonths[j].date.split("-");

    var aY = Number(a[0]), aM = Number(a[1]);

    var bY = Number(b[0]), bM = Number(b[1]);

    var diff = (aY \* 12 + aM) - (bY \* 12 + bM);

    difs.push(diff);

  }

  var periodicidade = "Anual";

  if (difs.length) {

    var avg = 0;

    for (var k = 0; k < difs.length; k++) avg += difs[k];

    avg = avg / difs.length;

    if (avg <= 1.5) periodicidade = "Mensal";

    else if (avg <= 3.5) periodicidade = "Trimestral";

    else if (avg <= 7)   periodicidade = "Semestral";

    else                 periodicidade = "Anual";

  }

  return {

    dividendo: Number(last.amount) || 0,

    mes: mesNome,

    periodicidade: periodicidade

  };

}

/\*\* Nome do mês PT a partir do número 1..12 \*/

function mesNomePt(m) {

  var nomes = ["", "Janeiro","Fevereiro","Março","Abril","Maio","Junho","Julho","Agosto","Setembro","Outubro","Novembro","Dezembro"];

  var n = Number(m);

  if (n >= 1 && n <= 12) return nomes[n];

  return "n/A";

}

/\*\* Normaliza periodicidade para PT canónico \*/

function normalizaPeriodicidade(p) {

  var s = String(p || "").trim().toLowerCase();

  if (!s) return "n/A";

  if (s === "mensal")     return "Mensal";

  if (s === "trimestral") return "Trimestral";

  if (s === "semestral")  return "Semestral";

  if (s === "anual")      return "Anual";

  var ignorar = ["na", "n/a", "n.d.", "não disponível", "n.a", "-"];

  if (ignorar.indexOf(s) >= 0) return "n/A";

  return "n/A";

}

/\*\* Normaliza mês (aceita nome PT, número, ou textos vazios) \*/

function normalizaMes(m) {

  if (m === null || m === undefined) return "n/A";

  var s = String(m).trim();

  if (!s) return "n/A";

  // número

  var n = Number(s);

  if (Number.isFinite(n) && n >= 1 && n <= 12) return mesNomePt(n);

  // nome já PT

  var nomes = ["janeiro","fevereiro","março","marco","abril","maio","junho","julho","agosto","setembro","outubro","novembro","dezembro","n/a","na","-"];

  var sLow = s.toLowerCase();

  if (nomes.indexOf(sLow) >= 0) {

    if (sLow === "marco") return "Março";

    if (sLow === "n/a" || sLow === "na" || sLow === "-") return "n/A";

    return sLow.charAt(0).toUpperCase() + sLow.slice(1);

  }

  return "n/A";

}

/\*\* Converte para number seguro \*/

function toNumber(v) {

  var n = Number(v);

  return Number.isFinite(n) ? n : 0;

}

versão mais recente:  
/\*\*

\* Sincroniza a folha "Firebase" com a coleção Firestore "acoesDividendos".

\* - PATCH de todos os campos (nome, ticker, setor, mercado, valorStock, periodicidade, mes, dividendo, taxas, origem e quaisquer extras que definires).

\* - Para não-ETFs, tenta enriquecer `dividendo` via Alpha Vantage em batches (respeita rate limit).

\* - Para ETFs, usa sempre o valor da sheet (não chama Alpha).

\* - Se o doc não existir, faz CREATE completo.

\*

\* Como adicionar novos campos?

\* 1) Vai à constante FIELD\_SPEC e adiciona um objeto com:

\* - name: nome do campo NO FIRESTORE (ex.: "peRatio")

\* - type: "string" | "number" | "boolean" | "null" (gera o tipo correto para Firestore)

\* - aliases: array com nomes/variações que podem aparecer no cabeçalho da Sheet

\* - required (opcional): se true, o script lança erro se o cabeçalho não existir

\* 2) (Opcional) Se precisares de normalização, acrescenta lógica em `normalizeValueByName`.

\*

\* NOTA: As colunas podem mudar de ordem e podem existir colunas novas; a correspondência é sempre por "alias" de cabeçalho.

\*/

// ======= Config =======

const USE\_ALPHA = true; // ativa/desativa Alpha Vantage (para não-ETFs)

const BATCH\_SIZE = 25; // nº de tickers enriquecidos por execução (Alpha)

const ALPHA\_SLEEP\_MS = 12000; // 12s entre chamadas -> ~5/min (plano gratuito)

const PROP\_KEY\_OFFSET = 'ALPHA\_OFFSET'; // offset rotativo para o batching

const ALPHA\_PROP\_KEY = 'ALPHA\_VANTAGE\_KEY';// nome da chave nas Script Properties

// ======= Logging =======

const VERBOSE = true; // ← põe false se quiseres “silencioso”

const LOG\_EVERY = 1; // logar cada N tickers (1 = todos)

// ==========================================================

// 1) ESPECIFICAÇÃO DE CAMPOS (Sheet -> Firestore)

// (adiciona aqui novos campos sem mexer no resto)

// ==========================================================

/\*\*

\* Cada item descreve 1 campo a ler da Sheet e a gravar no Firestore.

\* - name : Nome do campo no Firestore

\* - type : "string" | "number" | "boolean" | "null" (mapeia p/ stringValue, doubleValue, booleanValue, nullValue)

\* - aliases : Variações de cabeçalho possíveis na Sheet (case/acentos/esp espaços não importam)

\* - required: Se true, lança erro se o cabeçalho não existir

\*

\* 🟩 ADICIONA AQUI os teus novos campos com aliases correspondentes na Sheet.

\*/

const FIELD\_SPEC = [

// ——— CAMPO-CHAVE (obrigatórios) ———

{ name: "nome", type: "string", required: true, aliases: ["nome"] },

{ name: "ticker", type: "string", required: true, aliases: ["ticker"] },

// ——— Metadados principais ———

{ name: "setor", type: "string", aliases: ["setor","sector"] },

{ name: "mercado", type: "string", aliases: ["mercado","market"] },

{ name: "valorStock", type: "number", aliases: ["valorstock","preco","preço","preco atual","preço atual","price","valor stock","valor da ação"] },

// ——— Dividendos ———

{ name: "periodicidade", type: "string", aliases: ["periodicidade","periodicity"] },

{ name: "mes", type: "string", aliases: ["mes","mês","month"] },

{ name: "dividendo", type: "number", required: true, aliases: ["dividendo","dividend"] },

// ——— Taxas de crescimento ———

{ name: "taxaCrescimento\_1semana", type: "number", aliases: ["taxacrescimento\_1semana","taxa crescimento 1 semana","1w","1 semana"] },

{ name: "taxaCrescimento\_1mes", type: "number", aliases: ["taxacrescimento\_1mes","taxa crescimento 1 mes","taxa crescimento 1 mês","1m","1 mês","1 mes"] },

{ name: "taxaCrescimento\_1ano", type: "number", aliases: ["taxacrescimento\_1ano","taxa crescimento 1 ano","1y","1 ano"] },

// ——— 🟩 EXTRAS (exemplos; podes ativar/adaptar) ———

{ name: "peRatio", type: "number", aliases: ["p/e ratio (preço/lucro)","p/e ratio","pe","p e","p-l","p l","pe ratio"] },

{ name: "sma50", type: "number", aliases: ["sma50","sma 50","media móvel 50","média móvel 50","sma-50"] },

{ name: "sma200", type: "number", aliases: ["sma200","sma 200","media móvel 200","média móvel 200","sma-200"] },

{ name: "observacao",type: "string", aliases: ["observação","observacao","observations","obs","observ"] },

];

// ==========================================================

// Helpers gerais

// ==========================================================

/\*\* Normaliza strings para comparação de cabeçalho \*/

function norm(s){

return String(s||"")

.normalize("NFD").replace(/\p{Diacritic}/gu,"")

.replace(/\s+/g," ")

.trim().toLowerCase();

}

/\*\* Converte JS value para Firestore {xxxValue} conforme type \*/

function toFirestoreValue(type, value){

if (type === "number") {

const n = Number(value);

return Number.isFinite(n) ? { doubleValue: n } : { nullValue: null };

}

if (type === "boolean") {

return { booleanValue: !!value };

}

if (type === "null") {

return { nullValue: null };

}

// default string

return { stringValue: String(value ?? "") };

}

/\*\* Algumas normalizações por nome de campo \*/

function normalizeValueByName(name, rawValue){

if (name === "periodicidade") return normalizaPeriodicidade(rawValue);

if (name === "mes") return normalizaMes(rawValue);

if (name === "valorStock"

|| name === "dividendo"

|| name === "taxaCrescimento\_1semana"

|| name === "taxaCrescimento\_1mes"

|| name === "taxaCrescimento\_1ano"

|| name === "peRatio"

|| name === "sma50"

|| name === "sma200") {

const n = Number(rawValue);

return Number.isFinite(n) ? n : 0;

}

// string por omissão

return rawValue ?? "";

}

/\*\* Mapeia cabeçalhos da Sheet para índices de coluna, com base no FIELD\_SPEC \*/

function buildHeaderIndexMap(sheet){

const lastRow = sheet.getLastRow();

const lastColumn = sheet.getLastColumn();

if (lastRow < 2) return null;

const headerRaw = sheet.getRange(1, 1, 1, lastColumn).getValues()[0];

const headerNorm = headerRaw.map(norm);

const idx = {};

FIELD\_SPEC.forEach(spec => {

const found = headerNorm.findIndex(h => spec.aliases.map(norm).includes(h));

if (found < 0 && spec.required) {

throw new Error(`Cabeçalho obrigatório em falta para "${spec.name}". Esperado um de: ${spec.aliases.join(", ")}`);

}

idx[spec.name] = found; // -1 se não achou, permitido quando não-required

});

return { idx, lastRow, lastColumn };

}

/\*\* Heurística simples para detetar ETFs via nome/setor \*/

function isETF(nome, setor) {

var n = String(nome || "").toLowerCase();

var s = String(setor || "").toLowerCase();

if (s.startsWith("etf")) return true;

if (n.includes("etf")) return true;

if (n.includes("ucits")) return true;

if (n.includes("index fund")) return true;

if (n.includes("trust")) return true; // p.ex. QQQ Trust

return false;

}

/\*\* Nome do mês PT a partir do número 1..12 \*/

function mesNomePt(m) {

var nomes = ["", "Janeiro","Fevereiro","Março","Abril","Maio","Junho","Julho","Agosto","Setembro","Outubro","Novembro","Dezembro"];

var n = Number(m);

return (n >= 1 && n <= 12) ? nomes[n] : "n/A";

}

/\*\* Normaliza periodicidade para PT canónico \*/

function normalizaPeriodicidade(p) {

var s = String(p || "").trim().toLowerCase();

if (!s) return "n/A";

if (s === "mensal") return "Mensal";

if (s === "trimestral") return "Trimestral";

if (s === "semestral") return "Semestral";

if (s === "anual") return "Anual";

var ignorar = ["na", "n/a", "n.d.", "não disponível", "n.a", "-"];

if (ignorar.indexOf(s) >= 0) return "n/A";

return "n/A";

}

/\*\* Normaliza mês (aceita nome PT, número, ou textos vazios) \*/

function normalizaMes(m) {

if (m === null || m === undefined) return "n/A";

var s = String(m).trim();

if (!s) return "n/A";

var n = Number(s);

if (Number.isFinite(n) && n >= 1 && n <= 12) return mesNomePt(n);

var nomes = ["janeiro","fevereiro","março","marco","abril","maio","junho","julho","agosto","setembro","outubro","novembro","dezembro","n/a","na","-"];

var sLow = s.toLowerCase();

if (nomes.indexOf(sLow) >= 0) {

if (sLow === "marco") return "Março";

if (["n/a","na","-"].includes(sLow)) return "n/A";

return sLow.charAt(0).toUpperCase() + sLow.slice(1);

}

return "n/A";

}

/\*\* Converte para number seguro \*/

function toNumber(v) {

var n = Number(v);

return Number.isFinite(n) ? n : 0;

}

// ======= Helpers de logging =======

function logVerbose(msg){

if (VERBOSE) Logger.log(msg);

}

function logProgress(i, total, ticker, msg){

if (!VERBOSE) return;

if (LOG\_EVERY <= 1 || (i % LOG\_EVERY === 0)) {

Logger.log(`▶️ [${i}/${total}] ${ticker} — ${msg}`);

}

}

// ==========================================================

// 2) Funções Alpha Vantage (apenas para dividendos de não-ETFs)

// ==========================================================

function getDividendDataAlphaSafe(ticker, apiKey) {

var MAX\_TRIES = 3;

var lastErr = null;

for (var i = 0; i < MAX\_TRIES; i++) {

try {

return getDividendDataAlpha(ticker, apiKey);

} catch (e) {

lastErr = e;

Utilities.sleep(1500 \* (i + 1)); // backoff 1.5s, 3s

}

}

if (lastErr) throw lastErr;

return null;

}

function getDividendDataAlpha(ticker, apiKey) {

var key = apiKey || PropertiesService.getScriptProperties().getProperty(ALPHA\_PROP\_KEY);

if (!key) return null;

var url = "https://www.alphavantage.co/query?function=TIME\_SERIES\_MONTHLY\_ADJUSTED&symbol=" + encodeURIComponent(ticker) + "&apikey=" + encodeURIComponent(key);

var resp = UrlFetchApp.fetch(url, { method: "get", muteHttpExceptions: true });

var code = resp.getResponseCode();

if (code !== 200) throw new Error("Alpha HTTP " + code + " para " + ticker);

var json = JSON.parse(resp.getContentText());

var series = json["Monthly Adjusted Time Series"];

if (!series) return null;

// últimos meses com dividendos > 0

var months = Object.keys(series).sort().reverse(); // datas "YYYY-MM"

var divMonths = [];

for (var i = 0; i < months.length; i++) {

var mKey = months[i];

var dAmt = Number(series[mKey]["7. dividend amount"] || 0);

if (dAmt > 0) divMonths.push({ date: mKey, amount: dAmt });

if (divMonths.length >= 6) break; // suficiente para heurística

}

if (divMonths.length === 0) return null;

// Último dividendo

var last = divMonths[0];

var lastMonthNum = Number(last.date.split("-")[1]); // "YYYY-MM"

var mesNome = mesNomePt(lastMonthNum);

// Heurística de periodicidade

var difs = [];

for (var j = 1; j < divMonths.length; j++) {

var a = divMonths[j - 1].date.split("-");

var b = divMonths[j].date.split("-");

var aY = Number(a[0]), aM = Number(a[1]);

var bY = Number(b[0]), bM = Number(b[1]);

var diff = (aY \* 12 + aM) - (bY \* 12 + bM);

difs.push(diff);

}

var periodicidade = "Anual";

if (difs.length) {

var avg = difs.reduce((x,y)=>x+y,0) / difs.length;

if (avg <= 1.5) periodicidade = "Mensal";

else if (avg <= 3.5) periodicidade = "Trimestral";

else if (avg <= 7) periodicidade = "Semestral";

else periodicidade = "Anual";

}

return {

dividendo: Number(last.amount) || 0,

mes: mesNome,

periodicidade: periodicidade

};

}

// ==========================================================

// 3) Enviar para Firestore (CREATE/PATCH completo) + LOGGING

// ==========================================================

function enviarParaFirebase() {

var ss = SpreadsheetApp.getActiveSpreadsheet();

var sheet = ss.getSheetByName("Firebase");

if (!sheet) {

Logger.log("❌ Folha 'Firebase' não foi encontrada.");

return;

}

const map = buildHeaderIndexMap(sheet);

if (!map) {

Logger.log("⚠️ Sem dados para processar.");

return;

}

const data = sheet.getRange(2, 1, map.lastRow - 1, map.lastColumn).getValues();

const headers = { "Authorization": "Bearer " + ScriptApp.getOAuthToken() };

const totalLinhas = data.length;

let processed = 0, created = 0, updated = 0, skipped = 0, alphaHits = 0, alphaMiss = 0;

// 1) Lê documentos existentes → map por TICKER

var firestoreUrl = "https://firestore.googleapis.com/v1/projects/appfinance-812b2/databases/(default)/documents/acoesDividendos?pageSize=1000";

var response = UrlFetchApp.fetch(firestoreUrl, { method: "get", headers: headers, muteHttpExceptions: true });

if (response.getResponseCode() !== 200) {

Logger.log("❌ Erro ao listar Firestore: " + response.getContentText());

return;

}

var docs = JSON.parse(response.getContentText());

var firestoreDocs = docs.documents || [];

var firestoreMap = {}; // TICKER(UPPER) -> { docName, fields }

firestoreDocs.forEach(function(doc) {

var fields = doc.fields || {};

var tk = fields.ticker && fields.ticker.stringValue ? String(fields.ticker.stringValue).toUpperCase() : "";

if (tk) firestoreMap[tk] = { docName: doc.name, fields: fields };

});

// 2) Preparar janela/batching para Alpha (apenas não-ETFs)

var props = PropertiesService.getScriptProperties();

var alphaKey = props.getProperty(ALPHA\_PROP\_KEY);

var useAlphaThisRun = USE\_ALPHA && !!alphaKey;

var elegiveis = [];

for (var r0 = 0; r0 < data.length; r0++) {

// obtém valores mínimos p/ detetar ETF

const nome0 = getValueFromRow("nome", data[r0], map.idx);

const setor0 = getValueFromRow("setor", data[r0], map.idx);

const tk0raw = getValueFromRow("ticker", data[r0], map.idx);

const tk0 = String(tk0raw || "").trim().toUpperCase();

if (!tk0) continue;

if (!isETF(nome0, setor0)) elegiveis.push(r0);

}

var offset = parseInt(props.getProperty(PROP\_KEY\_OFFSET) || '0', 10);

if (!Number.isFinite(offset) || offset < 0 || offset >= Math.max(elegiveis.length,1)) offset = 0;

var windowLen = Math.min(BATCH\_SIZE, elegiveis.length);

var alphaIdxSet = new Set();

for (var iwin = 0; iwin < windowLen; iwin++) {

var idx = elegiveis[(offset + iwin) % Math.max(elegiveis.length,1)];

alphaIdxSet.add(idx);

}

logVerbose(`🚀 Início: ${totalLinhas} linhas para processar (VERBOSE=${VERBOSE})`);

// 3) Loop principal linha a linha

for (var r = 0; r < data.length; r++) {

var row = data[r];

// 3.1) Extrai TODOS os campos conforme FIELD\_SPEC

const record = {}; // { name: normalizedValue }

FIELD\_SPEC.forEach(spec => {

const raw = getValueFromRow(spec.name, row, map.idx);

record[spec.name] = normalizeValueByName(spec.name, raw);

});

// valida mínimos

const nome = String(record.nome || "").trim();

const ticker = String(record.ticker || "").trim().toUpperCase();

if (!nome || !ticker) {

skipped++;

Logger.log("⚠️ Linha ignorada (nome/ticker vazio). Linha " + (r + 2));

continue;

}

processed++;

logProgress(processed, totalLinhas, ticker, "a processar…");

// 3.2) Enriquecer dividendo via Alpha (apenas não-ETF + dentro da janela desta execução)

let finalDividendo = toNumber(record.dividendo);

const eETF = isETF(record.nome, record.setor);

if (eETF) {

logProgress(processed, totalLinhas, ticker, "ETF → não consulta Alpha; usa valor da Sheet");

} else if (useAlphaThisRun && alphaIdxSet.has(r)) {

logProgress(processed, totalLinhas, ticker, "Alpha: a consultar…");

try {

const alpha = getDividendDataAlphaSafe(ticker, alphaKey);

if (alpha && typeof alpha.dividendo === "number") {

finalDividendo = toNumber(alpha.dividendo);

alphaHits++;

logProgress(processed, totalLinhas, ticker, `Alpha OK → dividendo=${finalDividendo}`);

// Se quiseres alinhar também mês/periodicidade, descomenta:

// record.mes = alpha.mes || record.mes;

// record.periodicidade = alpha.periodicidade || record.periodicidade;

} else {

alphaMiss++;

logProgress(processed, totalLinhas, ticker, "Alpha sem dados de dividendo; mantém Sheet");

}

} catch (e) {

alphaMiss++;

logProgress(processed, totalLinhas, ticker, `Alpha ERRO: ${e}`);

}

Utilities.sleep(ALPHA\_SLEEP\_MS); // respeita rate limit

} else {

logProgress(processed, totalLinhas, ticker, "Alpha: fora da janela deste ciclo; mantém Sheet");

}

record.dividendo = finalDividendo;

// 3.3) Monta payload Firestore dinâmico (inclui "origem")

const payloadFields = {};

Object.keys(record).forEach(name => {

const spec = FIELD\_SPEC.find(s => s.name === name);

if (!spec) return; // safety

payloadFields[name] = toFirestoreValue(spec.type, record[name]);

});

// campo fixo "origem"

payloadFields.origem = { stringValue: "sheets" };

// 3.4) CREATE se não existir; caso exista, PATCH completo

const existing = firestoreMap[ticker];

try {

if (!existing) {

const createUrl = "https://firestore.googleapis.com/v1/projects/appfinance-812b2/databases/(default)/documents/acoesDividendos?documentId=" + encodeURIComponent(ticker);

const createResp = UrlFetchApp.fetch(createUrl, {

method: "post",

contentType: "application/json",

payload: JSON.stringify({ fields: payloadFields }),

headers: headers,

muteHttpExceptions: true

});

if (createResp.getResponseCode() >= 200 && createResp.getResponseCode() < 300) {

created++;

logProgress(processed, totalLinhas, ticker, "📄 Criado");

} else {

skipped++;

logProgress(processed, totalLinhas, ticker, `❌ Falha ao criar: ${createResp.getContentText()}`);

}

} else {

const updateUrl = "https://firestore.googleapis.com/v1/" + existing.docName;

const updateResp = UrlFetchApp.fetch(updateUrl, {

method: "patch",

contentType: "application/json",

payload: JSON.stringify({ fields: payloadFields }),

headers: headers,

muteHttpExceptions: true

});

if (updateResp.getResponseCode() >= 200 && updateResp.getResponseCode() < 300) {

updated++;

logProgress(processed, totalLinhas, ticker, "🔄 Atualizado");

} else {

skipped++;

logProgress(processed, totalLinhas, ticker, `❌ Falha ao atualizar: ${updateResp.getContentText()}`);

}

}

} catch (e) {

skipped++;

logProgress(processed, totalLinhas, ticker, `❌ Erro: ${e}`);

}

}

// 4) Avança offset da janela Alpha

if (elegiveis.length > 0) {

var nextOffset = (offset + windowLen) % elegiveis.length;

props.setProperty(PROP\_KEY\_OFFSET, String(nextOffset));

logVerbose(`⏭️ Próximo offset Alpha: ${nextOffset} / ${elegiveis.length}`);

} else {

logVerbose("ℹ️ Sem não-ETFs elegíveis para Alpha nesta execução");

}

logVerbose(`✅ Concluído: processados=${processed}, criados=${created}, atualizados=${updated}, ignorados/erros=${skipped}, alphaOK=${alphaHits}, alphaFalhas=${alphaMiss}`);

}

// ==========================================================

// Utilitário: obtém o valor de um campo numa row usando o map de índices

// ==========================================================

function getValueFromRow(fieldName, row, idxMap){

const col = idxMap[fieldName];

if (col === undefined || col === null || col < 0) return ""; // não encontrado (não required)

return row[col];

}

## explicação:

**🎯 Objetivo do script**

O código serve para **sincronizar a folha "Firebase" do Google Sheets com a coleção acoesDividendos do Firestore**. Ou seja: cada linha da sheet (ações/ETFs com dividendos) é enviada para o Firestore, sendo **criada ou atualizada** automaticamente.

**🔑 O que ele faz passo a passo**

1. **Configuração inicial**

* Decide se vai usar a **API Alpha Vantage** (USE\_ALPHA=true) para enriquecer dividendos (só para ações, nunca ETFs).
* Define batch (25 tickers por execução) e tempo de espera (12s) para não ultrapassar limites da Alpha Vantage.

1. **Ler dados da sheet**

* Abre a folha "Firebase", lê todas as linhas da coluna A..K (11 colunas).
* Se não encontrar nada, termina.

1. **Ver Firestore**

* Vai buscar a lista atual de documentos em acoesDividendos.
* Cria um mapa ticker → documento para saber se já existe ou não.

1. **Preparar enriquecimento Alpha**

* Identifica quais tickers **não são ETFs** (função isETF).
* Seleciona apenas uma “janela” de 25 tickers (rotativos, via offset guardado em propriedades) para chamar a Alpha Vantage nesta execução.

1. **Loop pelas linhas da sheet**

* Extrai os dados (nome, ticker, setor, mercado, valor stock, periodicidade, mês, dividendo, taxas de crescimento).
* Se não tiver ticker ou nome, ignora.
* **Se não for ETF e estiver na janela Alpha**, tenta enriquecer o valor do dividendo chamando a Alpha Vantage:
* Vai buscar os últimos dividendos pagos.
* Descobre mês e periodicidade (Mensal, Trimestral, etc.) por heurística.
* Cria um **payload completo** para o Firestore com todos os campos normalizados.
* Se o ticker **não existe no Firestore** → faz CREATE.
* Se já existe → faz PATCH (atualiza todos os campos).

1. **Gerir offset**

* Avança o offset (para que na próxima execução pegue nos próximos tickers).
* Assim, com várias execuções, todos os tickers acabam por ser enriquecidos via Alpha.

**⚙ Funções auxiliares**

* **isETF** → deteta ETFs pelo nome ou setor (contém ETF, UCITS, Trust…).
* **buildFullPayload** → formata os dados num objeto Firestore ({stringValue, doubleValue…}).
* **getDividendDataAlphaSafe** → busca dividendos na Alpha Vantage, com 3 tentativas e backoff.
* **getDividendDataAlpha** → consulta API, devolve último dividendo, mês e periodicidade.
* **normalizaPeriodicidade** / **normalizaMes** → padroniza texto da sheet (ex.: “marco” → “Março”).
* **toNumber** → garante que tudo que vai como número é válido (senão vira 0).

**Essência do algoritmo**

* **Lê a sheet → enriquece dividendos com Alpha (quando aplicável) → envia para Firestore (create/patch).**
* **Batch rotativo** garante não ultrapassar limite da API Alpha.
* **ETFs** nunca chamam Alpha, ficam apenas com dados da sheet.
* Mantém sempre os dados no Firestore **100% alinhados com a sheet**.

# *importarFirebaseParaSheet*

## código:

/\*\*

 \* Importa a coleção Firestore "acoesDividendos" para a folha "Firebase".

 \* - Atualiza/insere Nome, Ticker, Setor, Mercado, Periodicidade, Mês, Dividendo.

 \* - NÃO substitui fórmulas de E (valorStock) nem I/J/K (crescimentos).

 \* - Reaplica fórmulas de I/J/K se estiverem vazias (linhas existentes).

 \*/

function importarFirebaseParaSheet() {

  const ss = SpreadsheetApp.getActiveSpreadsheet();

  const sheet = ss.getSheetByName("Firebase");

  if (!sheet) {

    Logger.log("❌ A folha 'Firebase' não foi encontrada.");

    return;

  }

  const headers = { Authorization: 'Bearer ' + ScriptApp.getOAuthToken() };

  // Lê até 1000 docs (se precisares de mais, convém paginar)

  const firestoreUrl = 'https://firestore.googleapis.com/v1/projects/appfinance-812b2/databases/(default)/documents/acoesDividendos?pageSize=1000';

  const response = UrlFetchApp.fetch(firestoreUrl, { method: 'get', headers, muteHttpExceptions: true });

  if (response.getResponseCode() !== 200) {

    Logger.log("❌ Erro a obter Firestore: " + response.getContentText());

    return;

  }

  const firestoreDocs = (JSON.parse(response.getContentText()).documents || []);

  // Mapa de tickers já existentes na sheet (para atualizar no sítio certo)

  const lastRow = sheet.getLastRow();

  const existingRowsCount = Math.max(0, lastRow - 1);

  const data = existingRowsCount ? sheet.getRange(2, 1, existingRowsCount, 11).getValues() : [];

  const tickersMapSheet = {};

  data.forEach((row, i) => {

    const ticker = row[1];

    if (ticker) {

      tickersMapSheet[ticker] = { rowIndex: i + 2, rowData: row };

    }

  });

  firestoreDocs.forEach(doc => {

    const fields = doc.fields || {};

    const ticker = fields.ticker?.stringValue;

    if (!ticker) return;

    // Linha vinda do Firestore (sem fórmulas)

    const novaLinha = [

      fields.nome?.stringValue || "",       // A: nome

      ticker,                                // B: ticker

      fields.setor?.stringValue || "",       // C: setor

      fields.mercado?.stringValue || "",     // D: mercado

      "",                                    // E: valorStock (FÓRMULA — não escrevemos aqui)

      fields.periodicidade?.stringValue || "", // F

      fields.mes?.stringValue || "",         // G

      fields.dividendo?.doubleValue || 0     // H

      // I, J, K são fórmulas → não vêm do Firebase

    ];

    if (tickersMapSheet[ticker]) {

      // Atualiza linha existente

      const { rowIndex, rowData } = tickersMapSheet[ticker];

      // Compara campos A-D (0..3) + F-H (5..7)

      const precisaAtualizar =

        novaLinha.slice(0, 4).some((val, idx) => val !== rowData[idx]) ||

        novaLinha.slice(5, 8).some((val, idx) => val !== rowData[idx + 5]);

      if (precisaAtualizar) {

        sheet.getRange(rowIndex, 1, 1, 4).setValues([[novaLinha[0], novaLinha[1], novaLinha[2], novaLinha[3]]]);

        sheet.getRange(rowIndex, 6, 1, 3).setValues([[novaLinha[5], novaLinha[6], novaLinha[7]]]);

        // 👉 Repor fórmulas em I/J/K se estiverem vazias

        const valsIJK = sheet.getRange(rowIndex, 9, 1, 3).getValues()[0];

        const precisaFormulas =

          (valsIJK[0] === "" || valsIJK[0] === null) ||

          (valsIJK[1] === "" || valsIJK[1] === null) ||

          (valsIJK[2] === "" || valsIJK[2] === null);

        if (precisaFormulas) {

          preencherFormulasDeCrescimento(sheet, rowIndex, 1);

        }

        Logger.log(`🔄 Atualizado na Sheet: ${ticker}`);

      } else {

        Logger.log(`✅ Já estava atualizado: ${ticker}`);

      }

    } else {

      // Inserir NOVA linha

      const newRowIndex = sheet.getLastRow() + 1;

      // A..D

      sheet.getRange(newRowIndex, 1, 1, 4)

           .setValues([[novaLinha[0], novaLinha[1], novaLinha[2], novaLinha[3]]]);

      // F..H

      sheet.getRange(newRowIndex, 6, 1, 3)

           .setValues([[novaLinha[5], novaLinha[6], novaLinha[7]]]);

      // E: valorStock (fórmula). Mantive o teu exemplo com conversão USD→EUR.

      // Usa ';' como separador para bater com a locale da tua folha.

      sheet.getRange(newRowIndex, 5)

           .setFormula(`=IFERROR(ROUND(GOOGLEFINANCE(B${newRowIndex};"price")\*GOOGLEFINANCE("CURRENCY:USDEUR");2),"")`);

      // I/J/K: fórmulas de crescimento (só define se vazias)

      preencherFormulasDeCrescimento(sheet, newRowIndex, 1);

      Logger.log(`⬇️ Importado novo: ${ticker}`);

    }

  });

  Logger.log("✅ Importação da Firebase concluída.");

}

/\*\*

 \* Aplica fórmulas de crescimento (%) em:

 \*  - I (1 semana), J (1 mês), K (1 ano)

 \* Base: preço atual em E (valorStock) e GOOGLEFINANCE(B; "price"; TODAY()-X) para histórico.

 \* Só escreve se a célula estiver vazia (não destrói dados existentes).

 \*

 \* @param {GoogleAppsScript.Spreadsheet.Sheet} sheet

 \* @param {number} startRow  - linha inicial (≥2)

 \* @param {number} numRows   - quantas linhas aplicar

 \*/

function preencherFormulasDeCrescimento(sheet, startRow, numRows) {

  if (!numRows || numRows < 1) numRows = 1;

  // Lê B (ticker) e I..K (para saber se estão vazias)

  const tickers = sheet.getRange(startRow, 2, numRows, 1).getValues(); // col B

  const ijkVals = sheet.getRange(startRow, 9, numRows, 3).getValues(); // cols I..K

  for (let i = 0; i < numRows; i++) {

    const row = startRow + i;

    // Fórmulas com separador ';' (compatível com a tua folha)

    const fSem = `=IFERROR((E${row}/INDEX(GOOGLEFINANCE(B${row};"price";TODAY()-7);2;2)-1)\*100,"")`;

    const fMes = `=IFERROR((E${row}/INDEX(GOOGLEFINANCE(B${row};"price";TODAY()-30);2;2)-1)\*100,"")`;

    const fAno = `=IFERROR((E${row}/INDEX(GOOGLEFINANCE(B${row};"price";TODAY()-365);2;2)-1)\*100,"")`;

    // I

    if (ijkVals[i][0] === "" || ijkVals[i][0] === null) {

      sheet.getRange(row, 9).setFormula(fSem);

    }

    // J

    if (ijkVals[i][1] === "" || ijkVals[i][1] === null) {

      sheet.getRange(row, 10).setFormula(fMes);

    }

    // K

    if (ijkVals[i][2] === "" || ijkVals[i][2] === null) {

      sheet.getRange(row, 11).setFormula(fAno);

    }

  }

}

## explicação:

🎯 Objetivo

**Importar** a coleção **acoesDividendos** do Firestore para a folha **Firebase** no Google Sheets:

* **Atualiza ou insere**: Nome, Ticker, Setor, Mercado, Periodicidade, Mês, Dividendo.
* **Não toca** nas colunas com **fórmulas**: **E (valorStock)** e **I/J/K** (crescimentos 1 semana / 1 mês / 1 ano).
* **Repõe** as fórmulas de **I/J/K** **apenas se estiverem vazias**.

🧭 Passo a passo

1. **Abrir a folha e autenticar**

* Obtém a folha "Firebase".
* Prepara o cabeçalho HTTP com o token (ScriptApp.getOAuthToken()).

1. **Buscar documentos no Firestore**

* Faz um GET à API do Firestore (até **1000 docs**).
* Se falhar, regista o erro e para.

1. **Mapear linhas já existentes na sheet**

* Lê as linhas atuais (da linha 2 em diante) e cria um **mapa por ticker** → { ticker: { rowIndex, rowData } }.
* Isto permite **atualizar a linha certa** em vez de duplicar.

1. **Percorrer os documentos do Firestore**  
   Para cada doc:

* Extrai os campos (nome, ticker, setor, mercado, periodicidade, mes, dividendo).
* Monta um array **novaLinha** com os valores para as colunas **A..D** e **F..H**.
* **E (valorStock)** fica vazio aqui porque é **fórmula** na sheet.
* **Se o ticker já existe na sheet**:
* Compara A..D e F..H com o que está na linha.
* Se **mudou**, escreve:
* *A..D (nome, ticker, setor, mercado)*
* *F..H (periodicidade, mês, dividendo)*
* Depois **verifica I/J/K**. Se alguma estiver vazia, **repõe as fórmulas** com *preencherFormulasDeCrescimento*.
* **Se o ticker ainda não existe**:
* Insere **nova linha** no fim:
* Escreve A..D e F..H.
* Em **E (valorStock)**, coloca a **fórmula**:

=IFERROR(ROUND(GOOGLEFINANCE(B[row];"price")\*GOOGLEFINANCE("CURRENCY:USDEUR");2),"")

(pega o preço atual do ticker em USD e converte para EUR)

* Chama preencherFormulasDeCrescimento para **I/J/K**.

1. **Concluir**

* Escreve no log que terminou: "✅ Importação da Firebase concluída."

🧩 A função auxiliar: preencherFormulasDeCrescimento

* Aplica (apenas se a célula estiver vazia) as fórmulas:
* **I (1 semana)**  
  =IFERROR((E[row]/INDEX(GOOGLEFINANCE(B[row];"price";TODAY()-7);2;2)-1)\*100,"")
* **J (1 mês)**  
  =IFERROR((E[row]/INDEX(GOOGLEFINANCE(B[row];"price";TODAY()-30);2;2)-1)\*100,"")
* **K (1 ano)**  
  =IFERROR((E[row]/INDEX(GOOGLEFINANCE(B[row];"price";TODAY()-365);2;2)-1)\*100,"")
* As fórmulas calculam a **variação percentual** comparando o **valor atual (E)** com o **preço histórico** devolvido por GOOGLEFINANCE.

✅ Regras importantes que o script respeita

* **Não sobrescreve fórmulas** já existentes (E, I, J, K).
* **Só repõe fórmulas** de I/J/K se estiverem **vazias**.
* **Atualiza apenas** os campos de dados trazidos do Firestore (A..D, F..H).
* Evita duplicações usando um **mapa por ticker**.

📌 Limites e notas

* O GET ao Firestore está limitado a **pageSize=1000** (se tiveres mais docs, convém paginar).
* A fórmula de **E** assume **preço em USD** e converte para EUR — ajusta se o teu ticker já for em EUR.
* GOOGLEFINANCE pode falhar/atrasar; o IFERROR nas fórmulas evita quebrar a folha..

# *eliminarIncompletos*

## código:

function eliminarIncompletos() {

  const baseUrl = 'https://firestore.googleapis.com/v1/';

  const listUrl = baseUrl + 'projects/appfinance-812b2/databases/(default)/documents/acoesDividendos?pageSize=1000';

  const headers = {

    Authorization: 'Bearer ' + ScriptApp.getOAuthToken()

  };

  const response = UrlFetchApp.fetch(listUrl, { method: 'get', headers });

  const documentos = JSON.parse(response.getContentText()).documents || [];

  if (documentos.length === 0) {

    Logger.log("✅ Nenhum documento encontrado.");

    return;

  }

  documentos.forEach(doc => {

    const fields = doc.fields;

    const nome = fields?.nome?.stringValue || "";

    const setor = fields?.setor?.stringValue || "";

    const mercado = fields?.mercado?.stringValue || "";

    const mes = fields?.mes?.stringValue || "";

    const periodicidade = fields?.periodicidade?.stringValue || "";

    const valorStock = parseFloat(fields?.valorStock?.doubleValue);

    const estaIncompleto =

      nome === "" ||

      setor === "" ||

      mercado === "" ||

      mes === "" ||

      periodicidade === "" ||

      isNaN(valorStock);

    if (estaIncompleto) {

      const deleteUrl = baseUrl + doc.name;

      UrlFetchApp.fetch(deleteUrl, {

        method: 'delete',

        headers

      });

      Logger.log(`🗑️ Apagado da Firebase: ${fields?.ticker?.stringValue}`);

    }

  });

  Logger.log("✅ Limpeza da Firebase concluída.");

}

## explicação:

# *EliminarBDFirebase*

## código:

async function apagarTodosDocumentosFirebase() {

  const projectId = "appfinance-812b2"; // ⚠️ Altera se usares outro ID

  const urlBase = `https://firestore.googleapis.com/v1/projects/${projectId}/databases/(default)/documents/acoesDividendos`;

  const token = ScriptApp.getOAuthToken();

  // Obter todos os documentos da coleção

  const response = UrlFetchApp.fetch(urlBase, {

    method: "get",

    headers: {

      Authorization: `Bearer ${token}`

    },

    muteHttpExceptions: true

  });

  const dados = JSON.parse(response.getContentText());

  if (!dados.documents || dados.documents.length === 0) {

    Logger.log("❌ Nenhum documento encontrado na Firebase.");

    return;

  }

  let contador = 0;

  // Apagar cada documento individualmente

  dados.documents.forEach(doc => {

    const deleteUrl = `https://firestore.googleapis.com/v1/${doc.name}`;

    const deleteResponse = UrlFetchApp.fetch(deleteUrl, {

      method: "delete",

      headers: {

        Authorization: `Bearer ${token}`

      },

      muteHttpExceptions: true

    });

    if (deleteResponse.getResponseCode() === 200) {

      contador++;

      Logger.log(`✅ Documento apagado: ${doc.name}`);

    } else {

      Logger.log(`⚠️ Erro ao apagar: ${doc.name}`);

    }

  });

  Logger.log(`✅ ${contador} documentos apagados com sucesso.`);

}

## explicação:

# *Observação.gs*

## código:

/\*\*

 \* OBSERVACAO(price, pe, sma50, sma200, mom1m, [setor], [nome])

 \* - Aceita células únicas OU intervalos; ideal para ARRAYFORMULA.

 \* - Classifica: Subvalorizada (profunda), Subvalorizada, Sobrevalorizada (extrema),

 \*   Sobrevalorizada, Tendência alta/baixa, Justa, Potencial, Risco, Sem sinal.

 \*/

function OBSERVACAO(price, pe, sma50, sma200, mom1m, setorOpt, nomeOpt) {

  if (Array.isArray(price)) {

    const rows = price.length;

    const out = [];

    for (let i = 0; i < rows; i++) {

      out.push([ classify(

        pick(price,i), pick(pe,i), pick(sma50,i), pick(sma200,i), pick(mom1m,i),

        setorOpt ? pick(setorOpt,i) : "", nomeOpt ? pick(nomeOpt,i) : ""

      ) ]);

    }

    return out;

  }

  return classify(price, pe, sma50, sma200, mom1m, setorOpt || "", nomeOpt || "");

}

function pick(arg, i){ return Array.isArray(arg) ? (Array.isArray(arg[i]) ? arg[i][0] : arg[i]) : arg; }

function num(x){

  if (x===null || x===undefined) return NaN;

  if (typeof x==="number") return Number.isFinite(x)?x:NaN;

  const n = Number(String(x).trim().replace(",","."));

  return Number.isFinite(n)?n:NaN;

}

/\*\* Limiar por setor (podes ajustar à vontade) \*/

function getPEBands(setor, nome){

  const s = String(setor||"").toLowerCase();

  const n = String(nome||"").toLowerCase();

  // ETFs/Funds: ignorar P/E

  if (n.includes("etf") || n.includes("fund") || s.includes("etf")) {

    return { usePE:false, peCheap:NaN, peExpensive:NaN };

  }

  // Bancos/Financeiros (P/B seria melhor, mas com P/E usamos bandas mais baixas)

  if (s.includes("banco") || s.includes("finance")) return { usePE:true, peCheap:12, peExpensive:18 };

  // Energia/Telecom/Utilities/Materials/Industrial → P/E mais contido

  if (s.includes("energia") || s.includes("telecom") || s.includes("util") ||

      s.includes("materiais") || s.includes("indústr")) return { usePE:true, peCheap:14, peExpensive:22 };

  // Saúde/Farma

  if (s.includes("saúde") || s.includes("saude") || s.includes("farma")) return { usePE:true, peCheap:16, peExpensive:28 };

  // Tecnologia/Consumo Cíclico/Media

  if (s.includes("tecnologia") || s.includes("tech") || s.includes("consumo cíclico") || s.includes("consumo ciclico") ||

      s.includes("media") || s.includes("internet") ) return { usePE:true, peCheap:22, peExpensive:35 };

  // Default

  return { usePE:true, peCheap:18, peExpensive:30 };

}

function classify(price, pe, sma50, sma200, mom1m, setor, nome){

  const p   = num(price);

  const peN = num(pe);

  const s50 = num(sma50);

  const s200= num(sma200);

  const m1  = num(mom1m);

  if (!Number.isFinite(p)) return "";

  const has50  = Number.isFinite(s50);

  const has200 = Number.isFinite(s200);

  const hasPE  = Number.isFinite(peN);

  const { usePE, peCheap, peExpensive } = getPEBands(setor, nome);

  const belowBoth = has50 && has200 && p < Math.min(s50, s200);

  const aboveBoth = has50 && has200 && p > Math.max(s50, s200);

  const uptrend   = has50 && has200 && p > s50 && s50 > s200 && m1 >= 5;

  const downtrend = has50 && has200 && p < s50 && s50 < s200 && m1 <= -5;

  const deep = has200 && s200 > 0 && (p - s200)/s200 <= -0.12; // 12% abaixo da SMA200

  const belowOne = (has50 && !has200 && p < s50) || (has200 && !has50 && p < s200);

  const aboveOne = (has50 && !has200 && p > s50) || (has200 && !has50 && p > s200);

  // Caríssima independente do momentum (salvo se estiver muito abaixo da SMA200)

  if (usePE && hasPE && peN >= 60 && !(has200 && (p - s200)/s200 <= -0.20)) {

    return "Sobrevalorizada (extrema)";

  }

  // SUBVALORIZADA

  const cheapByPE = usePE && hasPE && peN <= peCheap;

  const cheapByMom = m1 >= 10; // forte momentum positivo é “válvula” pró-barato se preço < SMAs

  if ( (belowBoth || belowOne) && (cheapByPE || cheapByMom || !usePE) ) {

    return deep ? "Subvalorizada (profunda)" : "Subvalorizada";

  }

  // SOBREVALORIZADA

  const expensiveByPE = usePE && (!hasPE || peN >= peExpensive);

  if ( (aboveBoth || aboveOne) && expensiveByPE && m1 <= 0 ) {

    return "Sobrevalorizada";

  }

  // Tendências

  if (uptrend)   return "Tendência alta (pode crescer)";

  if (downtrend) return "Tendência baixa (atenção)";

  // Sem SMAs (ou ETFs sem PE): usar momentum + PE se existir

  if ((!has50 && !has200) || !usePE) {

    if (cheapByPE && m1 >= 5) return "Potencial (fund.+momentum)";

    if (usePE && hasPE && peN >= 30 && m1 <= 0) return "Risco de sobrevalorização";

    if (!usePE) { // ETF

      if (m1 >= 5)  return "ETF — momentum alto";

      if (m1 <= -5) return "ETF — sob pressão";

      return "ETF — neutro";

    }

    return "Sem sinal";

  }

  return "Justa";

}

## explicação:

O que a função faz

***OBSERVACAO(price, pe, sma50, sma200, mom1m, [setor], [nome])***  
Classifica uma ação (ou várias) como:

* **Subvalorizada (profunda)** / **Subvalorizada**
* **Sobrevalorizada (extrema)** / **Sobrevalorizada**
* **Tendência alta (pode crescer)** / **Tendência baixa (atenção)**
* **Justa**, **Potencial**, **Risco de sobrevalorização**
* **ETF — momentum alto/sob pressão/neutro** ou **Sem sinal**

Aceita **uma linha** ou **um intervalo inteiro** (útil com ARRAYFORMULA): se entra um intervalo, percorre cada linha e devolve uma coluna de classificações.

Entradas (variáveis)

* **price**: preço atual
* **pe**: P/E (Preço/Lucro) atual
* **sma50**: média móvel simples de 50 dias
* **sma200**: média móvel simples de 200 dias
* **mom1m**: “momentum” de 1 mês (variação % nos últimos ~30 dias)
* **setor** (opcional): usado para ajustar faixas “normais” de P/E por setor
* **nome** (opcional): usado para detectar **ETF**/**fund** (e então ignorar P/E)
* Há funções auxiliares que:
* **pick**: busca o valor certo quando as entradas são intervalos.
* **num**: normaliza/valida números (lida com vírgulas, vazios, etc.).

Bandas de P/E por setor (critério “barato/caro”)

A função **getPEBands(setor, nome)** devolve:

* usePE: se deve usar P/E (para **ETFs/fundos**, não usa)
* peCheap: limite abaixo do qual o P/E é considerado **barato**
* peExpensive: limite acima do qual é **caro**

Regras (podes ajustar):

* **Bancos/Financeiro**: barato ≤12 | caro ≥18
* **Energia/Telecom/Utilities/Materiais/Industrial**: barato ≤14 | caro ≥22
* **Saúde/Farma**: barato ≤16 | caro ≥28
* **Tecnologia/Consumo Cíclico/Média/Internet**: barato ≤22 | caro ≥35
* **Default**: barato ≤18 | caro ≥30
* **ETFs/Funds**: usePE = false (ignora P/E)

Regras de classificação (passo a passo)

1. **Limpeza & estado das SMAs/PE**

* Verifica se preço/SMAs/PE são numéricos.
* Calcula relações-chave:
* belowBoth: preço < **min(SMA50, SMA200)**
* aboveBoth: preço > **max(SMA50, SMA200)**
* **Tendência alta**: preço > SMA50 > SMA200 **e** momentum ≥ +5%
* **Tendência baixa**: preço < SMA50 < SMA200 **e** momentum ≤ −5%
* **Subvalorização profunda**: preço ≤ **−12%** abaixo da **SMA200**

1. **Sobrevalorização extrema por P/E**

* Se **P/E ≥ 60** ⇒ **“Sobrevalorizada (extrema)”**,  
  a menos que esteja **≥20% abaixo da SMA200** (mitiga “extrema” se preço já corrigiu muito).

1. **Subvalorizada**

* Se **preço abaixo** de uma ou ambas SMAs **e** (P/E **barato** ou momentum **≥ +10%** ou o ativo **não usa P/E**, p.ex. ETF), então:

Se também está ≥12% abaixo da SMA200 ⇒ **“Subvalorizada (profunda)”**

Caso contrário ⇒ **“Subvalorizada”**

1. **Sobrevalorizada**

* Se **preço acima** de uma ou ambas SMAs **e** P/E **caro** (ou em falta, mas com regra “caro”) **e** momentum **≤ 0%** ⇒ **“Sobrevalorizada”**

1. **Tendência**

* Se cumprir os encadeamentos:
* preço > SMA50 > SMA200 e momentum ≥ +5% ⇒ **“Tendência alta (pode crescer)”**
* preço < SMA50 < SMA200 e momentum ≤ −5% ⇒ **“Tendência baixa (atenção)”**

1. **Sem SMAs ou ETFs**

* Se não há SMAs **ou** é **ETF** (não usa P/E):
* momentum ≥ +5% e P/E barato ⇒ **“Potencial (fund.+momentum)”**
* P/E ≥ 30 e momentum ≤ 0 ⇒ **“Risco de sobrevalorização”**
* ETF sem sinal forte:
* momentum ≥ +5% ⇒ **“ETF — momentum alto”**
* momentum ≤ −5% ⇒ **“ETF — sob pressão”**
* caso contrário ⇒ **“ETF — neutro”**
* Se nada disso encaixa ⇒ **“Sem sinal”**

1. **Caso neutro**

* Se nenhuma condição acima disparar ⇒ **“Justa”**

Intuição por trás das regras

* **Preço vs. SMAs** indica o **regime técnico** (acima: força; abaixo: fraqueza).
* **P/E por setor** evita chamar “caro” o que é “normal” em tech, e “barato” o que é comum em bancos.
* **Momentum** (1 mês) atua como “válvula”: reforça tendência alta, atenua “barato” quando já há recuperação, e sinaliza cautela em “caro” com momentum fraco.
* **“Profunda”** usa a SMA200 como “linha de valor” de longo prazo (12% ou mais abaixo sugere desconto material).

Exemplos rápidos (mentais)

* **Preço abaixo de SMA50 e SMA200**, P/E abaixo da banda do setor ⇒ **Subvalorizada** (se ≥12% abaixo da SMA200 ⇒ **profunda**).
* **Preço acima das SMAs**, P/E acima da banda e momentum fraco ⇒ **Sobrevalorizada**.
* **Preço > SMA50 > SMA200** e momentum forte ⇒ **Tendência alta (pode crescer)**.
* **ETF** com momentum ≥ +5% ⇒ **ETF — momentum alto** (não olhamos P/E em ETFs).

Notas finais

* As **bandas de P/E** são **parametrizáveis** (podes afinar os limites por setor).
* mom1m espera **percentagem** (ex.: 7 significa **+7%** no mês).
* A função lida com **linhas em massa** (intervalos) e devolve uma coluna de rótulos correspondente.