```
/***** CONFIG *****/
const TZ = 'Europe/Madrid';
const SHEET_SOURCES = 'ICAL_SOURCES';  // PISO | ICS_URL
// En pruebas escribe en RESUMEN_TEST; en prod podrás cambiar arriba a
'RESUMEN'
const SHEET_RESUMEN = 'RESUMEN_TEST';


// Ventana temporal: SOLO próximos 30 días
const IMPORT_PAST_DAYS  = 0;
const IMPORT_FUTURE_DAYS = 30;


// Columnas mínimas en la hoja destino (si no existen, se añaden)
const COLUMNS_REQUIRED =
['Fecha','Tipo','Piso','Días','Estado','Clave','Origen','UID'];





/***** UTILIDADES *****/
function getSheet(name) {
  const sh = SpreadsheetApp.getActive().getSheetByName(name);
  if (!sh) throw new Error('Falta la hoja: ' + name);
  return sh;
}

function ensureColumns(sh, needed) {
  const lastCol = Math.max(1, sh.getLastColumn());
  const header = sh.getRange(1, 1, 1, lastCol).getValues()[0];
  const have = new Set(header.filter(Boolean));
  let col = header.length;
  needed.forEach(n => {
    if (!have.has(n)) {
      sh.getRange(1, ++col).setValue(n);
      header.push(n);
    }
  });
  return header;
}

function colIndexMap(header) {
  const m = {};
  header.forEach((h, i) => m[h] = i + 1);
  return m;
}

function asLocalDateStr(d) {
  return Utilities.formatDate(d, TZ, 'dd/MM/yyyy');
}

function inWindow(dt, from, to) {
  if (!dt) return false;
  const t = dt.getTime();
  return t >= from.getTime() && t <= to.getTime();
}

// —— NUEVAS utilidades comunes —— //
const _norm = s => String(s ?? '')
  .normalize('NFD').replace(/[\u0300-\u036f]/g,'')
  .toLowerCase().trim();

/** Convierte '58', '58,00 €', '27.800,00', '27.80' → número fiable */
function toNumber(v){
  if (typeof v === 'number') return v;
  if (v == null || v === '') return 0;
  let s = String(v).trim().replace(/[^\d,.\-]/g,'');
  const lastComma = s.lastIndexOf(','), lastDot = s.lastIndexOf('.');
  if (lastComma > lastDot) s = s.replace(/\./g,'').replace(',', '.');
  else s = s.replace(/,/g,'');
  const n = parseFloat(s);
  return Number.isFinite(n) ? n : 0;
}

/** Busca una columna por lista de nombres posibles (1-based). Devuelve 0
si no está. */
function findColByNames(header, names){
  const Hn = header.map(_norm);
  for (const name of (Array.isArray(names) ? names : [names])){
    const j = Hn.indexOf(_norm(name));
    if (j !== -1) return j+1;
  }
  return 0;
}


/***** PARSE ICS *****/
function parseDate(val){
  if (!val) return null;
  let s = String(val).trim();

  // TZID=Europe/Madrid:YYYY...
  if (s.includes(':')) s = s.split(':').pop();

  // ...CEST/CET al final
  const tzAbbr = s.match(/^(\d{8}T\d{6})([A-Z]{3,4})$/);
  if (tzAbbr) s = tzAbbr[1];

  // YYYYMMDD (al día)
  if (/^\d{8}$/.test(s)) {
    const d = new
Date(`${s.slice(0,4)}-${s.slice(4,6)}-${s.slice(6,8)}T00:00:00`);
    return isNaN(d.getTime()) ? null : d;
  }
  // YYYYMMDDTHHMMSSZ (UTC)
  if (/^\d{8}T\d{6}Z$/.test(s)) {
    const d = new
Date(`${s.slice(0,4)}-${s.slice(4,6)}-${s.slice(6,8)}T${s.slice(9,11)}:${s.
slice(11,13)}:${s.slice(13,15)}Z`);
    return isNaN(d.getTime()) ? null : d;
  }
  // YYYYMMDDTHHMMSS (local)
  if (/^\d{8}T\d{6}$/.test(s)) {
    const d = new
Date(`${s.slice(0,4)}-${s.slice(4,6)}-${s.slice(6,8)}T${s.slice(9,11)}:${s.
slice(11,13)}:${s.slice(13,15)}`);
```

```
    return isNaN(d.getTime()) ? null : d;
  }
  const d = new Date(s);
  return isNaN(d.getTime()) ? null : d;
}

function parseISODuration(dur) { // PnD
  if (!dur) return 0;
  const m = dur.match(/^P(\d+)D$/i);
  return m ? parseInt(m[1], 10) : 0;
}

function parseIcs(icsText) {
  const unfolded = icsText.replace(/\r\n[ \t]/g, ''); // RFC5545 unfold
  const blocks = unfolded.split('BEGIN:VEVENT').slice(1).map(b =>
'BEGIN:VEVENT' + b);

  return blocks.map(block => {
    const get = (prop) => {
      const m = block.match(new RegExp('^' + prop + '[:;](.+)$', 'm'));
      return m ? m[1].trim() : null;
    };

    const DTSTART = parseDate(get('DTSTART'));
    let DTEND = parseDate(get('DTEND'));
    const DURATION = get('DURATION'); // P3D, etc.

    if (!DTEND && DTSTART && DURATION) {
      const days = parseISODuration(DURATION);
      if (days > 0) {
        const tmp = new Date(DTSTART.getTime());
        tmp.setDate(tmp.getDate() + days);
        DTEND = tmp;
      }
    }
    if (!DTEND && DTSTART){
      const tmp = new Date(DTSTART.getTime());
      tmp.setDate(tmp.getDate() + 1); // estancia mínima 1 día
      DTEND = tmp;
    }

    return {
      uid: (get('UID') || '').trim(),
      summary: get('SUMMARY'),
      status: (get('STATUS') || '').toUpperCase(),
      dtstart: DTSTART,
      dtend: DTEND,
      allDay: /^\d{8}$/.test(get('DTSTART') || '')
    };
  });
}


/***** DEBUG *****/
function debugListSources() {
  const sh = getSheet(SHEET_SOURCES);
  const last = sh.getLastRow();
  if (last < 2) { Logger.log('No hay filas.'); return; }
  const rows = sh.getRange(2, 1, last - 1, 2).getValues();
  rows.forEach((r, i) => {
    const piso = r[0], url = r[1];
    Logger.log(`${i + 2}: ${piso || '(sin piso)'} | ${url ? 'URL OK' : 'SIN
URL'}`);
  });
}

function debugDumpFirstValidEvents(){
  const shSrc = getSheet(SHEET_SOURCES);
  const last = shSrc.getLastRow();
  if (last < 2){ Logger.log('No hay filas en ICAL_SOURCES'); return; }
  let rowIdx = null, piso=null, url=null;
  for (let r=2; r<=last; r++){
    const v = shSrc.getRange(r,1,1,2).getValues()[0];
    if (v[0] && v[1]){ rowIdx=r; piso=v[0]; url=v[1]; break; }
  }
  if (!rowIdx){ Logger.log('No hay ninguna fila válida con PISO+ICS_URL');
return; }
  const resp = fetchIcsRobusto(url);
  Logger.log(`Probando fila ${rowIdx}: ${piso} | HTTP
${resp.getResponseCode()}`);
  if (resp.getResponseCode() !== 200){ Logger.log('Respuesta no 200');
return; }
  const events = parseIcs(resp.getContentText());
  Logger.log(`Eventos totales: ${events.length}`);
  const shDbg = SpreadsheetApp.getActive().getSheetByName('DEBUG_EVENTS')
|| SpreadsheetApp.getActive().insertSheet('DEBUG_EVENTS');
  shDbg.clear();

shDbg.getRange(1,1,1,5).setValues([['UID','DTSTART','DTEND','STATUS','ALLDA
Y']]);
  const rows = events.slice(0,50).map(ev => [
    ev.uid,
    toIsoSafe(ev.dtstart),
    toIsoSafe(ev.dtend),
    ev.status || '',
    ev.allDay ? 'YES' : 'NO'
  ]);
  if (rows.length) shDbg.getRange(2,1,rows.length,5).setValues(rows);
}
function toIsoSafe(d){
  return (d && d.getTime && !isNaN(d.getTime())) ? d.toISOString() : '';
}


/***** FETCH ROBUSTO *****/
function fetchIcsRobusto(url){
  let resp = UrlFetchApp.fetch(url, { muteHttpExceptions: true,
followRedirects: true });
  if (resp.getResponseCode() === 200 && resp.getContentText().trim())
return resp;
  const sep = url.includes('?') ? '&' : '?';
  const busted = url + sep + '_cb=' + Date.now();
  resp = UrlFetchApp.fetch(busted, { muteHttpExceptions: true,
followRedirects: true });
  return resp;
}
```

```
/***** SYNC A RESUMEN_TEST *****/
function syncSalidas_TEST_allSources() {
  const shSrc = getSheet(SHEET_SOURCES);
  const shRes = getSheet(SHEET_RESUMEN);
  const header = ensureColumns(shRes, COLUMNS_REQUIRED);
  const H = colIndexMap(header);

  const now = new Date();
  const from = new Date(now); from.setDate(from.getDate() -
IMPORT_PAST_DAYS);
  const to   = new Date(now); to.setDate(to.getDate() +
IMPORT_FUTURE_DAYS);

  const last = shSrc.getLastRow();
  if (last < 2) { Logger.log('No hay fuentes.'); return; }
  const rows = shSrc.getRange(2, 1, last - 1, 2).getValues().filter(r =>
r[0] && r[1]);

  // Claves existentes -> fila
  const lastRes = shRes.getLastRow();
  const claveToRow = {};
  if (lastRes >= 2 && H['Clave']) {
    const claves = shRes.getRange(2, H['Clave'], lastRes - 1,
1).getValues().flat();
    claves.forEach((c, i) => { if (c) claveToRow[c] = i + 2; });
  }

  let totalNew = 0, totalUpd = 0;
  const bulkAppend = [];

  rows.forEach(([piso, url]) => {
    try {
      const resp = fetchIcsRobusto(url);
      if (resp.getResponseCode() !== 200) { Logger.log(`${piso}: HTTP
${resp.getResponseCode()}`); return; }
      const events = parseIcs(resp.getContentText());

      events.forEach(ev => {
        const { uid, status, dtend } = ev;
        if (!uid || !dtend) return;
        if (!inWindow(dtend, from, to)) return;

        const fechaStr = asLocalDateStr(dtend);          // SALIDA = DTEND
        const clave = `${piso}__${fechaStr}__SALIDA`;

        const registro = {
          'Fecha':  fechaStr,
          'Tipo':   'Limpieza salida',
          'Piso':   piso,
          'Días':   1,
          'Estado': status === 'CANCELLED' ? 'Cancelado' : 'Pendiente',
          'Clave':  clave,
          'Origen': 'KrossBooking',
          'UID':    uid
        };
        const payload = header.map(h => registro[h] ?? '');

        const row = claveToRow[clave];
        if (row) {
          shRes.getRange(row, 1, 1, header.length).setValues([payload]);
          totalUpd++;
        } else {
          bulkAppend.push(payload);
          claveToRow[clave] = (shRes.getLastRow() + bulkAppend.length);
          totalNew++;
        }
      });

    } catch (e) {
      Logger.log(`Error en ${piso}: ${e}`);
    }
  });

  if (bulkAppend.length) {
    const startRow = Math.max(2, shRes.getLastRow() + 1);
    shRes.getRange(startRow, 1, bulkAppend.length,
header.length).setValues(bulkAppend);
```

```
  }
  Logger.log(`Total nuevas: ${totalNew}. Total actualizadas:
${totalUpd}.`);
}


/***** COPIA/ENRIQUECE A RESUMEN *****/
function copyAndEnrichToResumen(){
  const SRC_NAME = 'RESUMEN_TEST';
  const DST_NAME = 'Resumen';
  const BASE_NAME = 'BASE_DATOS';

  const shSrc  = getSheet(SRC_NAME);
  const shDst  = getSheet(DST_NAME);
  const shBase = SpreadsheetApp.getActive().getSheetByName(BASE_NAME);

  const fmt = (d,pat='dd/MM/yyyy HH:mm:ss') => Utilities.formatDate(d, TZ,
pat);
  const toDateStr = (v)=> v instanceof Date && !isNaN(v) ?
Utilities.formatDate(v, TZ, 'dd/MM/yyyy') : String(v||'').trim();

  // Cabeceras
  const srcHeader =
shSrc.getRange(1,1,1,Math.max(1,shSrc.getLastColumn())).getValues()[0];
  let   dstHeader =
shDst.getRange(1,1,1,Math.max(1,shDst.getLastColumn())).getValues()[0];
  if (!srcHeader.length || !dstHeader.length) throw new Error('Revisa
encabezados en RESUMEN_TEST/RESUMEN');

  // Asegurar columnas destino mínimas + extras canónicas
  const mustHave =
['Fecha','Piso','Tipo','Días','Estado','Clave','Origen','UID','Marca de
Tiempo','Total'];
  mustHave.forEach(c=>{
    if (!dstHeader.includes(c)){
      dstHeader.push(c);
      shDst.getRange(1, dstHeader.length).setValue(c);
    }
  });
  // refrescar cabecera tras posibles altas
  dstHeader =
shDst.getRange(1,1,1,Math.max(1,shDst.getLastColumn())).getValues()[0];

  // Índices importantes (1-based)
  const iDFecha = findColByNames(dstHeader,'fecha');
  const iDPiso  = findColByNames(dstHeader,'piso');
  const iDTipo  = findColByNames(dstHeader,'tipo');
  const iDEstado= findColByNames(dstHeader,'estado');
  const iDClave = findColByNames(dstHeader,'clave');
  const iDOrigen= findColByNames(dstHeader,'origen');
  const iDUID   = findColByNames(dstHeader,'uid');
  const iDTs    = findColByNames(dstHeader,'marca de tiempo');
  const iDTotal = findColByNames(dstHeader,['total','total €','importe
total','total factura','total a facturar','total (eur)']);

  // Columnas a sumar para Total (excluye la columna Total por ÍNDICE)
  const sumIdx0 = dstHeader
    .map((h,j)=>({h:String(h||''), j}))
    .filter(o => /(precio|importe)/i.test(o.h) && (o.j !== (iDTotal-1)))
    .map(o => o.j); // 0-based

  const iSFecha = srcHeader.indexOf('Fecha') + 1;
  const iSPiso  = srcHeader.indexOf('Piso')  + 1;
  const iSEst   = srcHeader.indexOf('Estado')+ 1;
  const iSUID   = srcHeader.indexOf('UID')   + 1;
  if (iSFecha<=0 || iSPiso<=0) throw new Error('RESUMEN_TEST necesita
"Fecha" y "Piso".');

  // BASE_DATOS → mapa por Piso
  const baseMap = {};
  if (shBase){
    const baseHeader =
shBase.getRange(1,1,1,Math.max(1,shBase.getLastColumn())).getValues()[0];
    const lastB = shBase.getLastRow();
    if (lastB >= 2){
      const allB =
shBase.getRange(2,1,lastB-1,baseHeader.length).getValues();
      const iBPiso = baseHeader.findIndex(h => _norm(h)==='piso');
      allB.forEach(row=>{
        const piso = iBPiso>=0 ? row[iBPiso] : null;
        if (!piso) return;
        const rec = {};
        baseHeader.forEach((h,j)=>{ rec[String(h).trim()] = row[j]; });
        baseMap[String(piso).trim()] = rec;
      });
    }
  }

  // Mapa de existentes en RESUMEN (clave Piso__Fecha__tipo)
  const existing = new Map();
  const lastDst = shDst.getLastRow();
  if (lastDst >= 2){
    const rng = shDst.getRange(2,1,lastDst-1,dstHeader.length).getValues();
    rng.forEach((row, i)=>{
      const f = toDateStr(row[iDFecha-1]);
      const p = String(row[iDPiso-1]||'').trim();
      const t = iDTipo ? String(row[iDTipo-1]||'').trim().toLowerCase() :
'limpieza salida';
      if (f && p) existing.set(`${p}__${f}__${t}`, i+2);
    });
  }

  // Lectura de RESUMEN_TEST
  const lastSrc = shSrc.getLastRow();
  if (lastSrc < 2){ Logger.log('No hay datos nuevos en RESUMEN_TEST');
return; }
  const srcVals =
shSrc.getRange(2,1,lastSrc-1,shSrc.getLastColumn()).getValues();

  const nowStamp = fmt(new Date());
  const outRows = [];
  let add=0, upd=0, updCancel=0;

  const computeTotal = (rowArr) => sumIdx0.length
    ? sumIdx0.reduce((acc,j)=> acc + toNumber(rowArr[j]), 0) || ''
    : '';
  srcVals.forEach(r=>{
    const fecha = toDateStr(r[iSFecha-1]);
    const piso  = String(r[iSPiso-1]||'').trim();
    const est   = iSEst>0 ? String(r[iSEst-1]||'') : '';
    const uid   = iSUID>0 ? String(r[iSUID-1]||'') : '';
    if (!fecha || !piso) return;

    const tipo = 'Limpieza salida';
    const key  = `${piso}__${fecha}__${tipo.toLowerCase()}`;

    // Cancelaciones: marcar solo si existe
    if (_norm(est).startsWith('cancel')){
      const rowNum = existing.get(key);
      if (rowNum){
        if (iDEstado) shDst.getRange(rowNum,
iDEstado).setValue('Cancelado');
        if (iDTs)     shDst.getRange(rowNum, iDTs).setValue(nowStamp);
        updCancel++;
      }
      return;
    }

    // Construir payload del tamaño de la cabecera destino
    const payload = new Array(dstHeader.length).fill('');
    payload[iDFecha-1] = fecha;
    payload[iDPiso-1]  = piso;
    if (iDTipo)   payload[iDTipo-1]   = tipo;
    if (iDEstado) payload[iDEstado-1] = 'Pendiente';
    if (iDClave)  payload[iDClave-1]  = `${piso}__${fecha}__SALIDA`;
    if (iDOrigen) payload[iDOrigen-1] = 'KrossBooking';
    if (iDUID && uid) payload[iDUID-1] = uid;
    if (iDTs)     payload[iDTs-1]     = nowStamp;

    // Enriquecer desde BASE_DATOS por nombre de columna
    const rec = baseMap[piso];
    if (rec){
      Object.keys(rec).forEach(col=>{
        const pos = dstHeader.indexOf(col);
        if (pos>=0 && (payload[pos]==='' || payload[pos]==null))
payload[pos] = rec[col];
      });
    }

    const rowExisting = existing.get(key);
    if (rowExisting){
      const current =
shDst.getRange(rowExisting,1,1,dstHeader.length).getValues()[0];
      const merged = current.map((v,j)=>{
        if (dstHeader[j]==='Clave' && v) return v;           // conserva
Clave anterior
        return (v==='' || v==null) ? payload[j] : v;
      });
      if (iDOrigen) merged[iDOrigen-1] = 'KrossBooking';
      if (iDUID && uid) merged[iDUID-1] = uid;
      if (iDTs) merged[iDTs-1] = nowStamp;
      if (iDTotal) merged[iDTotal-1] = computeTotal(merged);

      shDst.getRange(rowExisting,1,1,dstHeader.length).setValues([merged]);
      upd++;
    } else {
      if (iDTotal) payload[iDTotal-1] = computeTotal(payload);
      outRows.push(payload);
      existing.set(key, (shDst.getLastRow() + outRows.length));
      add++;
    }
  });

  if (outRows.length){
    const startRow = Math.max(2, shDst.getLastRow()+1);

shDst.getRange(startRow,1,outRows.length,dstHeader.length).setValues(outRow
s);
  }
  // Formato de moneda para toda la columna Total (si existe)
  if (iDTotal && shDst.getLastRow() >= 2){
    shDst.getRange(2, iDTotal, shDst.getLastRow()-1, 1).setNumberFormat('€
#,##0.00');
```

```
  }

  Logger.log(`RESUMEN <- RESUMEN_TEST: nuevas ${add}, actualizadas ${upd},
canceladas ${updCancel}.`);
}


/***** ORQUESTADOR + TRIGGER *****/
function runKrossEvery2h(){
  syncSalidas_TEST_allSources(); // iCal → RESUMEN_TEST (próx. 30 días)
  copyAndEnrichToResumen();      // RESUMEN_TEST → RESUMEN (rellena y sin
duplicar; marca cancelados)
  pushResumenToCalendars();      // ← NUEVO: crea/borra en Google Calendar
(stub)
}

function createTrigger_KrossEvery2h(){
  ScriptApp.newTrigger('runKrossEvery2h')
    .timeBased()
    .everyHours(2)
    .create();
}



/***** RECÁLCULOS TOTAL *****/
function recalcTotalSoloColumna(){
  const SHEET = 'Resumen';
  const sh = SpreadsheetApp.getActive().getSheetByName(SHEET);
  if (!sh) throw new Error('No encuentro la hoja RESUMEN');

  let lastRow = sh.getLastRow();
  let lastCol = sh.getLastColumn();
  if (lastRow < 2) return;

  let header = sh.getRange(1,1,1,lastCol).getValues()[0];

  // Ubicar/crear Total
  let iTotal = findColByNames(header, ['total','total €','total
factura','total a facturar','importe total','total (eur)']);
  if (!iTotal){
    iTotal = lastCol + 1;
    sh.getRange(1, iTotal).setValue('Total');
    lastCol = iTotal;
    header  = sh.getRange(1,1,1,lastCol).getValues()[0];
  }

  // Columnas a sumar (excluir Total por índice)
  const sumIdx0 = header
    .map((h,j)=>({h:String(h||''), j}))
    .filter(o => /(precio|importe)/i.test(o.h) && (o.j !== (iTotal-1)))
    .map(o => o.j);

  if (!sumIdx0.length) throw new Error('No encuentro columnas con
"Precio"/"Importe" para sumar.');

  const rng = sh.getRange(2,1,lastRow-1,lastCol);
  const values = rng.getValues();

  const out = values.map(row => {
    const suma = sumIdx0.reduce((acc,j)=> acc + toNumber(row[j]), 0);
    return [suma || ''];
  });

  sh.getRange(2, iTotal, out.length, 1).setValues(out);
  sh.getRange(2, iTotal, out.length, 1).setNumberFormat('€ #,##0.00');

  Logger.log(`Total recalculado en ${out.length} filas. Sumo:
${sumIdx0.map(j=>header[j]).join(' + ')}`);
}

function recalcTotalHastaHoy(){
  const SHEET = 'Resumen';
  const sh = SpreadsheetApp.getActive().getSheetByName(SHEET);
  if (!sh) throw new Error('No encuentro la hoja RESUMEN');
  const lastRow = sh.getLastRow(), lastCol = sh.getLastColumn();
  if (lastRow < 2) return;
  const header = sh.getRange(1,1,1,lastCol).getValues()[0];
  const iFecha = findColByNames(header, 'fecha');
  if (!iFecha) throw new Error('Falta la columna "Fecha"');

  let iTotal = findColByNames(header, ['total','total €','importe
total','total factura','total a facturar','total (eur)']);
  if (!iTotal){
    iTotal = lastCol + 1;
    sh.getRange(1, iTotal).setValue('Total');
  }

  const sumIdx0 = header
    .map((h,j)=>({h:String(h||''), j}))
    .filter(o => /(precio|importe)/i.test(o.h) && (o.j !== (iTotal-1)))
    .map(o => o.j);

  const today0 = new Date(Utilities.formatDate(new Date(), TZ,
'yyyy/MM/dd') + ' 00:00:00');

  const toDate = (v)=>{
    if (v instanceof Date && !isNaN(v)) return v;
    if (typeof v === 'number'){ const base = new
Date(Date.UTC(1899,11,30)); return new Date(base.getTime()+v*86400000); }
    const s = String(v||'').trim();
    let m = s.match(/^(\d{1,2})[\/\-](\d{1,2})[\/\-](\d{2,4})/);
    if (m){ let [_,dd,mm,yy]=m; if(yy.length===2) yy='20'+yy; return new
Date(`${yy}-${mm.padStart(2,'0')}-${dd.padStart(2,'0')}T00:00:00`); }
    m = s.match(/^(\d{4})-(\d{1,2})-(\d{1,2})/);
    if (m){ const [_,yy,mm,dd]=m; return new
Date(`${yy}-${String(mm).padStart(2,'0')}-${String(dd).padStart(2,'0')}T00:
00:00`); }
    return null;
  };

  const rng = sh.getRange(2,1,lastRow-1,Math.max(lastCol,iTotal));
  const values = rng.getValues();

  let tocadas=0;
  for (let r=0;r<values.length;r++){
    const row = values[r];
    const fecha = toDate(row[iFecha-1]);
    if (!fecha || fecha > today0) continue;
    const total = sumIdx0.reduce((acc,j)=> acc + toNumber(row[j]), 0);
    row[iTotal-1] = total || '';
    values[r] = row; tocadas++;
  }
  if (tocadas){
    rng.setValues(values);
    sh.getRange(2, iTotal, values.length, 1).setNumberFormat('€ #,##0.00');
  }
  Logger.log(`Total recalculado hasta hoy en ${tocadas} filas.`);
}


/***** MENÚ *****/
function buildMenuResumenMensual(){
  SpreadsheetApp.getUi()
    .createMenu('RESUMEN • Utilidades')
    .addItem('Recalcular Total (todas)', 'recalcTotalSoloColumna')
    .addItem('Recalcular Total (hasta hoy)', 'recalcTotalHastaHoy')
    .addToUi();
}
```