PRIMER LABORATORIO

Andres Fabian Rueda Contreras

20211578059

Diseño y Análisis de un Sistema Lógico Versátil para Bases 2-1024

Universidad Distrital Francisco José de Caldas

Facultad Tecnológica

Profesor Gerardo Alberto Castang Montiel

Bogotá, Colombia

22 de agosto de 2023

Introducción:

En el ámbito de la informática y la electrónica digital, el diseño de sistemas que operan en múltiples bases numéricas es un desafío fundamental. Estos sistemas permiten realizar conversiones precisas y eficientes entre distintas bases, lo que resulta esencial para diversas aplicaciones, desde aritmética computacional hasta codificación de datos. El presente informe aborda el diseño y la implementación de un sistema que opera en un rango amplio de bases numéricas, específicamente desde la base 2 hasta la base 1024. Este sistema tiene la capacidad de realizar conversiones entre bases de origen y destino dentro de este rango, mientras cumple con condiciones específicas para garantizar la precisión y la eficiencia de las conversiones.

El sistema propuesto se desarrolla teniendo en cuenta la especificación de tres condiciones distintas. Primero, cuando no existe una relación directa entre las bases de origen y destino, se utiliza un enfoque de conversión en dos etapas: primero se convierte el número de la base de origen a base 10 y luego a la base de destino. Segundo, cuando hay una relación directa de potencias entre las bases, se aplica un método de potencias para lograr la conversión directa sin pasar por la base 10. Finalmente, como requisito esencial, se verifica que el número en base 10 antes y después de la conversión sea el mismo, asegurando así la precisión de las operaciones.

Este informe detalla el proceso de diseño del sistema, su implementación en un lenguaje de programación y los resultados obtenidos a partir de pruebas exhaustivas. Asimismo, se presentan ejemplos concretos que ilustran las diferentes condiciones y cómo el sistema resuelve cada una de ellas.

Objetivo:

General:

Diseñar, implementar y evaluar un sistema de conversión numérica versátil y eficiente que opere en un rango amplio de bases numéricas, abarcando desde la base 2 hasta la base 1024

Específicos:

- Crear una base de datos con la cual convertir los números dados, esta deberá de tener un límite de 1024 caracteres.
- Desarrollar un diseño lógico que permita la conversión precisa entre bases numéricas dentro del rango de 2 a 1024.
- Implementar el diseño a un lenguaje de programación.
- Diseñar y ejecutar pruebas exhaustivas para verificar la precisión de las conversiones realizadas por el sistema.
- Presentar ejemplos concretos de conversiones entre bases numéricas, destacando diferentes escenarios en los que se aplican las condiciones establecidas.

Solución:

En primera instancia habría que crear una “base de datos” para saber con qué caracteres se trabajará en la conversión de bases. Se aplicó lo siguiente:

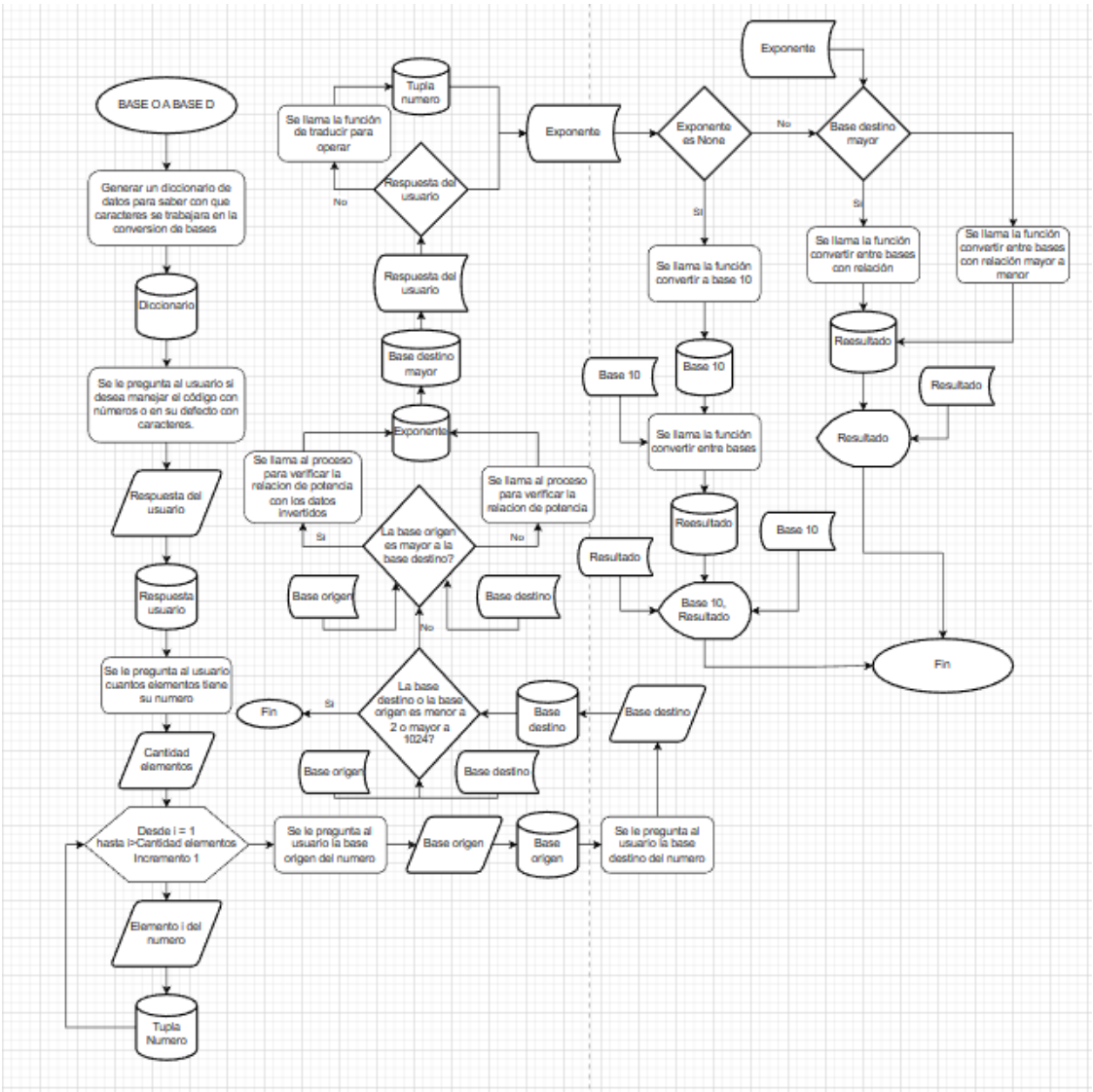

```
{0: '0', 1: '1', 2: '2', 3: '3', 4: '4', 5: '5', 6: '6', 7: '7', 8: '8', 9: '9', 10: 'A', 11: 'B', 12: 'C', 13: 'D', 14: 'E', 15: 'F', 16: 'G', 17: 'H', 18: 'I', 19:
 'J', 20: 'K', 21: 'L', 22: 'M', 23: 'N', 24: 'Ñ', 25: 'O', 26: 'P', 27: 'Q', 28: 'R', 29: 'S', 30: 'T', 31: 'U', 32: 'V', 33: 'W', 34: 'X', 35: 'Y', 36: 'Z', 37: 'a
', 38: 'b', 39: 'c', 40: 'd', 41: 'e', 42: 'f', 43: 'g', 44: 'h', 45: 'i', 46: 'j', 47: 'k', 48: 'l', 49: 'm', 50: 'n', 51: 'ñ', 52: 'o', 53: 'p', 54: 'q', 55: 'r',
56: 's', 57: 't', 58: 'u', 59: 'v', 60: 'w', 61: 'x', 62: 'y', 63: 'z', 64: '!A', 65: '!B', 66: '!C', 67: '!D', 68: '!E', 69: '!F', 70: '!G', 71: '!H', 72: '!I', 73:
 '!J', 74: '!K', 75: '!L', 76: '!M', 77: '!N', 78: '!Ñ', 79: '!O', 80: '!P', 81: '!Q', 82: '!R', 83: '!S', 84: '!T', 85: '!U', 86: '!V', 87: '!W', 88: '!X', 89: '!Y
', 90: '!Z', 91: '!a', 92: '!b', 93: '!c', 94: '!d', 95: '!e', 96: '!f', 97: '!g', 98: '!h', 99: '!i', 100: '!j', 101: '!k', 102: '!l', 103: '!m', 104: '!n', 105: '!ñ
', 106: '!o', 107: '!p', 108: '!q', 109: '!r', 110: '!s', 111: '!t', 112: '!u', 113: '!v', 114: '!w', 115: '!x', 116: '!y', 117: '!z', 118: '@A', 119: '@B', 120: '@C
', 121: '@D', 122: '@E', 123: '@F', 124: '@G', 125: '@H', 126: '@I', 127: '@J', 128: '@K', 129: '@L', 130: '@M', 131: '@N', 132: '@Ñ', 133: '@O', 134: '@P', 135: '@Q
', 136: '@R', 137: '@S', 138: '@T', 139: '@U', 140: '@V', 141: '@W', 142: '@X', 143: '@Y', 144: '@Z', 145: '@a', 146: '@b', 147: '@c', 148: '@d', 149: '@e', 150: '@f
', 151: '@g', 152: '@h', 153: '@i', 154: '@j', 155: '@k', 156: '@l', 157: '@m', 158: '@n', 159: '@ñ', 160: '@o', 161: '@p', 162: '@q', 163: '@r', 164: '@s', 165: '@t
', 166: '@u', 167: '@v', 168: '@w', 169: '@x', 170: '@y', 171: '@z', 172: '$A', 173: '$B', 174: '$C', 175: '$D', 176: '$E', 177: '$F', 178: '$G', 179: '$H', 180: '$I
', 181: '$J', 182: '$K', 183: '$L', 184: '$M', 185: '$N', 186: '$Ñ', 187: '$O', 188: '$P', 189: '$Q', 190: '$R', 191: '$S', 192: '$T', 193: '$U', 194: '$V', 195: '$W
', 196: '$X', 197: '$Y', 198: '$Z', 199: '$a', 200: '$b', 201: '$c', 202: '$d', 203: '$e', 204: '$f', 205: '$g', 206: '$h', 207: '$i', 208: '$j', 209: '$k', 210: '$l
', 211: '$m', 212: '$n', 213: '$ñ', 214: '$o', 215: '$p', 216: '$q', 217: '$r', 218: '$s', 219: '$t', 220: '$u', 221: '$v', 222: '$w', 223: '$x', 224: '$y', 225: '$z
', 226: '%A', 227: '%B', 228: '%C', 229: '%D', 230: '%E', 231: '%F', 232: '%G', 233: '%H', 234: '%I', 235: '%J', 236: '%K', 237: '%L', 238: '%M', 239: '%N', 240: '%Ñ
', 241: '%O', 242: '%P', 243: '%Q', 244: '%R', 245: '%S', 246: '%T', 247: '%U', 248: '%V', 249: '%W', 250: '%X', 251: '%Y', 252: '%Z', 253: '%a', 254: '%b', 255: '%c
', 256: '%d', 257: '%e', 258: '%f', 259: '%g', 260: '%h', 261: '%i', 262: '%j', 263: '%k', 264: '%l', 265: '%m', 266: '%n', 267: '%ñ', 268: '%o', 269: '%p', 270: '%q
', 271: '%r', 272: '%s', 273: '%t', 274: '%u', 275: '%v', 276: '%w', 277: '%x', 278: '%y', 279: '%z', 280: '^A', 281: '^B', 282: '^C', 283: '^D', 284: '^E', 285: '^F
', 286: '^G', 287: '^H', 288: '^I', 289: '^J', 290: '^K', 291: '^L', 292: '^M', 293: '^N', 294: '^Ñ', 295: '^O', 296: '^P', 297: '^Q', 298: '^R', 299: '^S', 300: '^T
', 301: '^U', 302: '^V', 303: '^W', 304: '^X', 305: '^Y', 306: '^Z', 307: '^a', 308: '^b', 309: '^c', 310: '^d', 311: '^e', 312: '^f', 313: '^g', 314: '^h', 315: '^i
', 316: '^j', 317: '^k', 318: '^l', 319: '^m', 320: '^n', 321: '^ñ', 322: '^o', 323: '^p', 324: '^q', 325: '^r', 326: '^s', 327: '^t', 328: '^u', 329: '^v', 330: '^w
', 331: '^x', 332: '^y', 333: '^z', 334: '&A', 335: '&B', 336: '&C', 337: '&D', 338: '&E', 339: '&F', 340: '&G', 341: '&H', 342: '&I', 343: '&J', 344: '&K', 345: '&L
', 346: '&M', 347: '&N', 348: '&Ñ', 349: '&O', 350: '&P', 351: '&Q', 352: '&R', 353: '&S', 354: '&T', 355: '&U', 356: '&V', 357: '&W', 358: '&X', 359: '&Y', 360: '&Z
', 361: '&a', 362: '&b', 363: '&c', 364: '&d', 365: '&e', 366: '&f', 367: '&g', 368: '&h', 369: '&i', 370: '&j', 371: '&k', 372: '&l', 373: '&m', 374: '&n', 375: '&ñ
', 376: '&o', 377: '&p', 378: '&q', 379: '&r', 380: '&s', 381: '&t', 382: '&u', 383: '&v', 384: '&w', 385: '&x', 386: '&y', 387: '&z', 388: '(A', 389: '(B', 390: '(C
', 391: '(D', 392: '(E', 393: '(F', 394: '(G', 395: '(H', 396: '(I', 397: '(J', 398: '(K', 399: '(L', 400: '(M', 401: '(N', 402: '(Ñ', 403: '(O', 404: '(P', 405: '(Q
', 406: '(R', 407: '(S', 408: '(T', 409: '(U', 410: '(V', 411: '(W', 412: '(X', 413: '(Y', 414: '(Z', 415: '(a', 416: '(b', 417: '(c', 418: '(d', 419: '(e', 420: '(f
', 421: '(g', 422: '(h', 423: '(i', 424: '(j', 425: '(k', 426: '(l', 427: '(m', 428: '(n', 429: '(ñ', 430: '(o', 431: '(p', 432: '(q', 433: '(r', 434: '(s', 435: '(t
', 436: '(u', 437: '(v', 438: '(w', 439: '(x', 440: '(y', 441: '(z', 442: ')A', 443: ')B', 444: ')C', 445: ')D', 446: ')E', 447: ')F', 448: ')G', 449: ')H', 450: ')I
', 451: ')J', 452: ')K', 453: ')L', 454: ')M', 455: ')N', 456: ')Ñ', 457: ')O', 458: ')P', 459: ')Q', 460: ')R', 461: ')S', 462: ')T', 463: ')U', 464: ')V', 465: ')W
', 466: ')X', 467: ')Y', 468: ')Z', 469: ')a', 470: ')b', 471: ')c', 472: ')d', 473: ')e', 474: ')f', 475: ')g', 476: ')h', 477: ')i', 478: ')j', 479: ')k', 480: ')l
', 481: ')m', 482: ')n', 483: ')ñ', 484: ')o', 485: ')p', 486: ')q', 487: ')r', 488: ')s', 489: ')t', 490: ')u', 491: ')v', 492: ')w', 493: ')x', 494: ')y', 495: ')z
', 496: '_A', 497: '_B', 498: '_C', 499: '_D', 500: '_E', 501: '_F', 502: '_G', 503: '_H', 504: '_I', 505: '_J', 506: '_K', 507: '_L', 508: '_M', 509: '_N', 510: '_Ñ
', 511: '_O', 512: '_P', 513: '_Q', 514: '_R', 515: '_S', 516: '_T', 517: '_U', 518: '_V', 519: '_W', 520: '_X', 521: '_Y', 522: '_Z', 523: '_a', 524: '_b', 525: '_c
', 526: '_d', 527: '_e', 528: '_f', 529: '_g', 530: '_h', 531: '_i', 532: '_j', 533: '_k', 534: '_l', 535: '_m', 536: '_n', 537: '_ñ', 538: '_o', 539: '_p', 540: '_q
', 526: '_d', 527: '_e', 528: '_f', 529: '_g', 530: '_h', 531: '_i', 532: '_j', 533: '_k', 534: '_l', 535: '_m', 536: '_n', 537: '_ñ', 538: '_o', 539: '_p', 540: '_q
', 541: '_r', 542: '_s', 543: '_t', 544: '_u', 545: '_v', 546: '_w', 547: '_x', 548: '_y', 549: '_z', 550: '=A', 551: '=B', 552: '=C', 553: '=D', 554: '=E', 555: '=F
', 556: '=G', 557: '=H', 558: '=I', 559: '=J', 560: '=K', 561: '=L', 562: '=M', 563: '=N', 564: '=Ñ', 565: '=O', 566: '=P', 567: '=Q', 568: '=R', 569: '=S', 570: '=T
', 571: '=U', 572: '=V', 573: '=W', 574: '=X', 575: '=Y', 576: '=Z', 577: '=a', 578: '=b', 579: '=c', 580: '=d', 581: '=e', 582: '=f', 583: '=g', 584: '=h', 585: '=i
', 586: '=j', 587: '=k', 588: '=l', 589: '=m', 590: '=n', 591: '=ñ', 592: '=o', 593: '=p', 594: '=q', 595: '=r', 596: '=s', 597: '=t', 598: '=u', 599: '=v', 600: '=w
', 601: '=x', 602: '=y', 603: '=z', 604: '{A', 605: '{B', 606: '{C', 607: '{D', 608: '{E', 609: '{F', 610: '{G', 611: '{H', 612: '{I', 613: '{J', 614: '{K', 615: '{L
', 616: '{M', 617: '{N', 618: '{Ñ', 619: '{O', 620: '{P', 621: '{Q', 622: '{R', 623: '{S', 624: '{T', 625: '{U', 626: '{V', 627: '{W', 628: '{X', 629: '{Y', 630: '{Z
', 631: '{a', 632: '{b', 633: '{c', 634: '{d', 635: '{e', 636: '{f', 637: '{g', 638: '{h', 639: '{i', 640: '{j', 641: '{k', 642: '{l', 643: '{m', 644: '{n', 645: '{ñ
', 646: '{o', 647: '{p', 648: '{q', 649: '{r', 650: '{s', 651: '{t', 652: '{u', 653: '{v', 654: '{w', 655: '{x', 656: '{y', 657: '{z', 658: '}A', 659: '}B', 660: '}C
', 661: '}D', 662: '}E', 663: '}F', 664: '}G', 665: '}H', 666: '}I', 667: '}J', 668: '}K', 669: '}L', 670: '}M', 671: '}N', 672: '}Ñ', 673: '}O', 674: '}P', 675: '}Q
', 676: '}R', 677: '}S', 678: '}T', 679: '}U', 680: '}V', 681: '}W', 682: '}X', 683: '}Y', 684: '}Z', 685: '}a', 686: '}b', 687: '}c', 688: '}d', 689: '}e', 690: '}f
', 691: '}g', 692: '}h', 693: '}i', 694: '}j', 695: '}k', 696: '}l', 697: '}m', 698: '}n', 699: '}ñ', 700: '}o', 701: '}p', 702: '}q', 703: '}r', 704: '}s', 705: '}t
', 706: '}u', 707: '}v', 708: '}w', 709: '}x', 710: '}y', 711: '}z', 712: '[A', 713: '[B', 714: '[C', 715: '[D', 716: '[E', 717: '[F', 718: '[G', 719: '[H', 720: '[I
', 721: '[J', 722: '[K', 723: '[L', 724: '[M', 725: '[N', 726: '[Ñ', 727: '[O', 728: '[P', 729: '[Q', 730: '[R', 731: '[S', 732: '[T', 733: '[U', 734: '[V', 735: '[W
', 736: '[X', 737: '[Y', 738: '[Z', 739: '[a', 740: '[b', 741: '[c', 742: '[d', 743: '[e', 744: '[f', 745: '[g', 746: '[h', 747: '[i', 748: '[j', 749: '[k', 750: '[l
', 751: '[m', 752: '[n', 753: '[ñ', 754: '[o', 755: '[p', 756: '[q', 757: '[r', 758: '[s', 759: '[t', 760: '[u', 761: '[v', 762: '[w', 763: '[x', 764: '[y', 765: '[z
', 766: ']A', 767: ']B', 768: ']C', 769: ']D', 770: ']E', 771: ']F', 772: ']G', 773: ']H', 774: ']I', 775: ']J', 776: ']K', 777: ']L', 778: ']M', 779: ']N', 780: ']Ñ
', 781: ']O', 782: ']P', 783: ']Q', 784: ']R', 785: ']S', 786: ']T', 787: ']U', 788: ']V', 789: ']W', 790: ']X', 791: ']Y', 792: ']Z', 793: ']a', 794: ']b', 795: ']c
', 796: ']d', 797: ']e', 798: ']f', 799: ']g', 800: ']h', 801: ']i', 802: ']j', 803: ']k', 804: ']l', 805: ']m', 806: ']n', 807: ']ñ', 808: ']o', 809: ']p', 810: ']q
', 811: ']r', 812: ']s', 813: ']t', 814: ']u', 815: ']v', 816: ']w', 817: ']x', 818: ']y', 819: ']z', 820: '?A', 821: '?B', 822: '?C', 823: '?D', 824: '?E', 825: '?F
', 826: '?G', 827: '?H', 828: '?I', 829: '?J', 830: '?K', 831: '?L', 832: '?M', 833: '?N', 834: '?Ñ', 835: '?O', 836: '?P', 837: '?Q', 838: '?R', 839: '?S', 840: '?T
', 841: '?U', 842: '?V', 843: '?W', 844: '?X', 845: '?Y', 846: '?Z', 847: '?a', 848: '?b', 849: '?c', 850: '?d', 851: '?e', 852: '?f', 853: '?g', 854: '?h', 855: '?i
', 856: '?j', 857: '?k', 858: '?l', 859: '?m', 860: '?n', 861: '?ñ', 862: '?o', 863: '?p', 864: '?q', 865: '?r', 866: '?s', 867: '?t', 868: '?u', 869: '?v', 870: '?w
', 871: '?x', 872: '?y', 873: '?z', 874: ':A', 875: ':B', 876: ':C', 877: ':D', 878: ':E', 879: ':F', 880: ':G', 881: ':H', 882: ':I', 883: ':J', 884: ':K', 885: ':L
', 886: ':M', 887: ':N', 888: ':Ñ', 889: ':O', 890: ':P', 891: ':Q', 892: ':R', 893: ':S', 894: ':T', 895: ':U', 896: ':V', 897: ':W', 898: ':X', 899: ':Y', 900: ':Z
', 901: ':a', 902: ':b', 903: ':c', 904: ':d', 905: ':e', 906: ':f', 907: ':g', 908: ':h', 909: ':i', 910: ':j', 911: ':k', 912: ':l', 913: ':m', 914: ':n', 915: ':ñ
', 916: ':o', 917: ':p', 918: ':q', 919: ':r', 920: ':s', 921: ':t', 922: ':u', 923: ':v', 924: ':w', 925: ':x', 926: ':y', 927: ':z', 928: '<A', 929: '<B', 930: '<C
', 931: '<D', 932: '<E', 933: '<F', 934: '<G', 935: '<H', 936: '<I', 937: '<J', 938: '<K', 939: '<L', 940: '<M', 941: '<N', 942: '<Ñ', 943: '<O', 944: '<P', 945: '<Q
', 946: '<R', 947: '<S', 948: '<T', 949: '<U', 950: '<V', 951: '<W', 952: '<X', 953: '<Y', 954: '<Z', 955: '<a', 956: '<b', 957: '<c', 958: '<d', 959: '<e', 960: '<f
', 961: '<g', 962: '<h', 963: '<i', 964: '<j', 965: '<k', 966: '<l', 967: '<m', 968: '<n', 969: '<ñ', 970: '<o', 971: '<p', 972: '<q', 973: '<r', 974: '<s', 975: '<t
', 976: '<u', 977: '<v', 978: '<w', 979: '<x', 980: '<y', 981: '<z', 982: '>A', 983: '>B', 984: '>C', 985: '>D', 986: '>E', 987: '>F', 988: '>G', 989: '>H', 990: '>I
', 991: '>J', 992: '>K', 993: '>L', 994: '>M', 995: '>N', 996: '>Ñ', 997: '>O', 998: '>P', 999: '>Q', 1000: '>R', 1001: '>S', 1002: '>T', 1003: '>U', 1004: '>V', 100
5: '>W', 1006: '>X', 1007: '>Y', 1008: '>Z', 1009: '>a', 1010: '>b', 1011: '>c', 1012: '>d', 1013: '>e', 1014: '>f', 1015: '>g', 1016: '>h', 1017: '>i', 1018: '>j',
1019: '>k', 1020: '>l', 1021: '>m', 1022: '>n', 1023: '>ñ', 1024: '>o', 1025: '>p', 1026: '>q', 1027: '>r', 1028: '>s', 1029: '>t', 1030: '>u', 1031: '>v', 1032: '>w
', 1033: '>x', 1034: '>y', 1035: '>z', 1036: '/A', 1037: '/B', 1038: '/C', 1039: '/D', 1040: '/E', 1041: '/F', 1042: '/G', 1043: '/H', 1044: '/I', 1045: '/J', 1046:
```

Se implementa lo aplicado en clase con la conversión de bases hasta la 64, para expandirlo hasta la base 1024 o al menos que la cubra se repite los caracteres de la base 10 hasta la 63 con un dígito de más obteniendo así una base de 1089 sin embargo se trabajara hasta la 1024.

Una vez finalizada la base de datos con la cual trabajar las conversiones de bases se empezará con el código.

Se decide trabajar con el lenguaje de programación python.

Para empezar con el código primero se tenía que ordenar las ideas y los procesos sobre cuál debería ir primero y la toma de decisiones durante el código, se implementó el siguiente diagrama de flujo para poder facilitar todo esto y programar con más certeza.



Una vez se obtiene un modelo a seguir se empieza la implementación en código:

```
diccionario = {}
dic_traducido = []
numeros=[]
resultado=[]
max = 27
min = 0
letrasMayus = {1:"A",2:"B",3:"C",4:"D",5:"E",6:"F",7:"G",8:"H",9:"I",10:"J",11:"K",12:"L",13:"M",14:"N",15:"Ñ",
    16:"O",17:"P",18:"Q",19:"R",20:"S",21:"T",22:"U",23:"V",24:"W",25:"X",26:"Y",27:"Z"}
letras = {1:"a",2:"b",3:"c",4:"d",5:"e",6:"f",7:"g",8:"h",9:"i",10:"j",11:"k",12:"l",13:"m",14:"n",15:"ñ",
    16:"o",17:"p",18:"q",19:"r",20:"s",21:"t",22:"u",23:"v",24:"w",25:"x",26:"y",27:"z"}
simbolos = {1:"!",2:"@",3:"$",4:"%",5:"^",6:"&",7:"(",8:")",9:"_",10:"=",11:"{",12:"}",13:"[",14:"]",15:"?",16:":",17:"<",18:">",19:"/"}
dic_digito =[]
dic_digito_numeros =[]
numero_destino = []
```

Se empieza por inicializar variables.

```
def asignar_base_datos():
    minn=min
    temp=0
    num= 1
    for temp in range (10): #Asigna los primeros 10 valores

        diccionario.update({temp:str(temp)})
    for minn in range (max): #Asigna los valores 10 al 36
        temp +=1
        diccionario.update({temp:letrasMayus.get(minn+1)})
    minn = min
    for minn in range (max): #Asigna los valores del 37 al 63
        temp +=1
        diccionario.update({temp:letras.get(minn+1)})
    minn = min
    for num in range (19): #Rellena los valores que faltan repitiendo los procesos pasados 20 veces mas
        for minn in range (max):
            temp +=1
            letra = simbolos.get(num+1)+letrasMayus.get(minn+1)
            diccionario.update({temp:letra})
        minn = min
        for minn in range (max):
            temp +=1
            letra = simbolos.get(num+1)+letras.get(minn+1)
            diccionario.update({temp:letra})
        minn = min
```

Y se llama al método asignar_base_datos() está lo que hace son ciclos repetitivos y en cada ciclo almacena la llave que sería el contador temp mas el caracter o símbolo respectivo que en su defecto serán los mencionados anteriormente.

```
def main():
    asignar_base_datos()
    respuesta = input("Manejar el codigo con numeros?")
    cantidad_elementos = input("Ingrese cuantos elementos tiene su numero: ")
    if int(cantidad_elementos)>0:
        for numeros in range (int(cantidad_elementos)):
            temp=input(f"Ingrese el digito #{numeros+1}: ")

            dic_digito.append(temp)
        base_origen = int(input("Ingrese la base origen (entre 2 y 1024): "))
        base_destino = int(input("Ingrese la base destino (entre 2 y 1024): "))
        if base_origen < 2 or base_origen > 1024 or base_destino < 2 or base_destino > 1024:
            print("Las bases deben estar en el rango de 2 a 1024.")
            return
```

Luego se le empiezan a pedir los datos al usuario, si desea manejar el código con símbolos o con números, cuántos elementos tiene el número la base origen y la base destino.

Cuando se le indica cuántos elementos tiene el número se realizará un ciclo para obtener dicha cantidad de elementos, estos elementos serán almacenados en una tupla dic_digito para próximamente usarlos.

```
    exponente = 0
    tupla_int = []
    son_numeros = False
    tempp=False
    if base_origen>base_destino:
        exponente = find_power_relation(base_destino, base_origen)
        tempp = False
    else:
        exponente = find_power_relation(base_origen, base_destino)
        tempp = True
    if respuesta == "y":
        son_numeros = True
        dic_digito_numeros = dic_digito
        tupla_int = tuple(int(item) for item in dic_digito_numeros)
        print(tupla_int)
    elif respuesta == "n":
        son_numeros = False
```

Posteriormente se verifica si la base origen es mayor a la base destino esto para que no existan errores a la hora de verificar si es una relación de potencias, también se guarda una variable temporal tempp para próximamente usarla en una caso de que exista relación de potencia y la base destino sea menor a la base origen. (traduce la tupla a entero)

si la base origen es mayor a la base destino se llamará a la funcion find_power_relation y se le enviaran los valores de base destino y base origen, en caso de que la base origen sea menor a la base destino se enviaran estos mismos valores pero invertidos es decir primero la base origen y luego base destino, se guardará el resultado de la función find_power_relation en una variable llamada exponente.

Luego se verifica si la respuesta del usuario a si quería usar el código con números es “y” o “n” para traducir o no el número suministrado, se guarda en una variable son_numeros la decisión del usuario.

```python
def find_power_relation(base_origen, base_destino):
    exponente = 0

    # Verificar si base_origen^exponent es igual a base_destino
    while base_origen ** exponente != base_destino:
        exponente += 1
        if base_origen ** exponente > base_destino:
            return None  # No hay relación de potencia

    return exponente
```

Inicializa una variable en 0 para luego realizar un ciclo while que verifica que la base_origen elevado al exponente sea distinta del destino, es decir mientras que sean distintas va a permanecer en el ciclo. dentro del ciclo aumenta 1 en el exponente y luego verifica si la operación anterior pasa a la base destino en este caso no habría una relación de potencia por lo tanto se retorna None (no hay relación de potencia) en caso contrario de que salga del ciclo ya que halla encontrado el exponente y la base_origen y la base_destino concuerden retornara el exponente.

```python
if son_numeros:
    print("Los valores estan en numeros")
    if exponente is None:
        print("NO relacion de potencia")
        resultado = convert_between_bases(tupla_int, base_origen, base_destino)
        print(f"Convirtiendo de base {base_origen} a base 10: {tupla_int} = {convert_to_base_10(tupla_int, base_origen)}")
        print(f"Convirtiendo a base {base_destino}: {tupla_int} = {resultado}")
```

Verifica si la variable son_numeros es verdadera, en dado caso imprime en pantalla la decisión tomada (Los valores estan en numeros) luego verifica si el exponente es None es decir si no hay una relación de potencia, si no hay relación de potencia imprime la decisión tomada (No relación de potencia) luego llama a la funcion convert_between_bases enviando la tupla del número la base origen y la base destino y guarda la respuesta de la función en una variable resultado.

llama una segunda función para que le retorne el valor en base 10 enviándole la tupla_int y la base_origen esta segunda función la imprime directamente. luego imprime el resultado de la función convert_between_bases

```
def convert_between_bases(num_str, base_origin, base_dest):
    if base_origin == base_dest:
        return num_str

    if base_origin == 10:
        num_in_base_dest = convert_from_base_10(num_str, base_dest)
    elif base_dest == 10:
        num_in_base_dest = convert_to_base_10(num_str, base_origin)
    else:
        num_in_base_10 = convert_to_base_10(num_str, base_origin)
        num_in_base_dest = convertir_from_base_10(num_in_base_10, base_dest)

    return num_in_base_dest
```

La funcion convert_between_bases recibe 3 parametros el número en tupla, la base origen y la base destino y retorna el número en la base destino. dentro de la función verifica si la base origen y la base destino son las mismas para retornar el mismo número suministrado ya que no necesita de conversión.

verifica si la base origen está en base 10 para llamar a la función convert_from_base_10 enviando el número y la base destino. el resultado de la función se guardará en num_in_base_dest

verifica si la base destino es base 10 para llamar a la función convert_to_base_10 enviando el número y la base origen. el resultado de la función se guardará en num_in_base_dest

si no entro en ningún if anterior lo convierte primero en base 10 con la función convert_to_base_10 para luego enviarle el resultado y la base destino a la función convert_from_base_10. el resultado de la función se guardará en num_in_base_dest. por último retorna num_in_base_dest.

```
def convert_from_base_10(number, base):
    print(number)
    numer = ''.join(str(digit) for digit in number)
    numero = int(numer)
    result = ""
    while numero > 0:
        digit = numero % base
        numero_destino.insert(0,digit)
        traducir = traduccion_digitos(digit)
        result = traducir+ result
        numero //= base
    return result
```

guarda la tupla en un solo número entero es decir si la tupla estaba en [1,2,3,4] se guardará en numer como 1234 y numero convierte a numer en entero. inicializa la variable result como un str ya que se va a retornar caracteres.

realiza un while mientras numero sea mayor a 0, en el ciclo calcula el residuo de la división del número numero por la base especificada base y lo guarda en digit. luego guarda el dígito en la primera posición de numero_destino. Se llama una función para traducir a dígitos enviandole el dígito y almacenando el resultado en traducir. luego guarda traducir en resultado pero lo guarda de una manera que traducir este siempre en la primera posición.

luego realiza la operación de división entera y reasignación en numero, asi ira desglosando a division guardando el residuo de cada división.

cuando no se pueda dividir más retornara el resultado.

```
def traduccion_digitos(digito):
    numero = diccionario.get(digito)
    return numero
```

En la función de traduccion_digitos simplemente hace una asignación buscando la llave de cada item, ya que se esta operando numeros y el diccionario tiene como claves números esta operación es bastante sencilla, retorna el numero que este caso no es un número sino un símbolo.

```python
def convert_to_base_10(num_str, base_ori):
    exponente = len(num_str)
    resultado_base_10 = 0
    for x in range(1, exponente+1):
        resultado_base_10 += num_str[x-1]*((base_ori)**(exponente-x))
    return resultado_base_10
```

En la función convert_to_base_10 se le suministra la tupa del número y la base origen, luego calcula la longitud de la tupla y la guarda en exponente, inicializa la variable resultado_base_10 en 0. luego realiza un for que permite iterar a través de cada dígito en la cadena. en cada iteración del bucle el código realiza lo siguiente:

Convierte el dígito actual (en la posición x-1) de la cadena num_str en un número entero.

luego calcula el valor de la base original elevada a la potencia correspondiente. Esto refleja la posición del dígito en la cadena, donde el dígito más a la izquierda es el dígito más significativo y el dígito más a la derecha es el menos significativo.

Multiplica el dígito por la potencia de la base correspondiente.

Agrega el resultado de la multiplicación al resultado_base_10. Esto acumula el valor calculado para cada dígito en base original en la variable resultado_base_10.

Al final retorna resultado_base_10

```python
def convertir_from_base_10(number, base):
    result = ""
    while number > 0:
        digit = number % base
        numero_destino.insert(0,digit)
        traducir = traduccion_digitos(digit)
        result = traducir+ result
        number //= base
    return result
```

Esta función realiza el mismo procedimiento que la de convert_from_base_10 pero con una excepción ya que esta función está modificada para que reciba un número entero ya que cuando se quiere pasar de una base diferente de 10 a otra base diferente de 10 realiza 2 funciones convert_to_base_10 y convert_from_base_10 la primera convierte el número a base 10 y retorna un número entero el problema recae ahí cuando se quiere manejar un número entero como una tupla ya que eso hace convert_from_base_10 sin embargo cree una segunda función con el mismo procedimiento a excepción que este esta modificado para operar con un número entero.

```
else:
    print("relacion de potencia")
    if tempp:
        print("normal")
        resultado = convert_between_bases_relacion(tupla_int, base_origen, base_destino)
        print(f"El número {tupla_int} en base {base_origen} es equivalente a {resultado} en base {base_destino}.")
    else:
        print("Invertido")
        resultado = rela_potencias_mayor_a_menor(base_destino,tupla_int, base_origen)
        print(f"El número {tupla_int} en base {base_origen} es equivalente a {resultado} en base {base_destino}.")
```

Si Existe una relacion de potencia se verificara la variable temporal anterior (tempp) para verificar si la base origen es mayor o menor en caso de que tempo sea verdadera se informara al usuario con “normal” asi que realizara la funcion convert_between_bases_relacion enviandole la tupla del numero la base origen y la base destino. luego se imprime en pantalla el resultado de esa funcion.

en caso contrario de que tempp sea falsa se le informara que esta invertido asi que realizara una funcion distinta para poder calcular el numero en la base solicitada. se llama la funcion rela_potencias_mayor_a_menor enviandole la base destino la tupla del numero y la base origen para finalizar imprimiendo el resultado de la funcion en pantalla. todo esto sin pasar por base 10.

```
def convert_between_bases_relacion(dic_digito, base_origen, base_destino):
    num_int_list = [int(digit) for digit in dic_digito]
    decimal_value = convert_to_decimal(num_int_list, base_origen)
    result = convert_to_base_n(decimal_value, base_destino)
    return result
```

En esta funcion se transforma la tupla dada en una lista para poder trabajar el numero mas facilmente. se llama a la funcion convert_to_decimal enviandole el numero en una lista y la base origen, esta funcion guarda el resultado en decimal_value. luego se llama a otra funcion llamada convert_to_base_n enviandole decimal_value y la base destino y guarda el resultado de la funcion en result. para finalizar retorna result.

```python
def convert_to_decimal(number, base_origen):
    numero = int(''.join(str(valor) for valor in number))
    decimal_value = 0
    power = 0
    while numero > 0:
        digit = numero % 10
        decimal_value += digit * (base_origen ** power)
        power += 1
        numero //= 10
    return decimal_value
```

En esta funcion convierte la lista en un numero entero, (ahora que me doy de cuenta converti la tupla a una lista para nada por que es la misma operacion que la tupla) inicializo dos variables decimal_value y power.
con un ciclo while verifico que el numero sea mayor a 0 para poder realizar lo siguiente: Empieza calculando el digito menos significativo de numero y lo guarda en digit. luego calcula el valor del dígito en la base original elevado a la potencia correspondiente incrementa 1 en power y luego elimina el digito menos significativo. esto se repite hasta que no hayan digitos en numero al finalizar retorna decimal_value.

```python
def convert_to_base_n(decimal_value, base_destino):
    if decimal_value == 0:
        return "0"

    result = ""
    while decimal_value > 0:
        remainder = decimal_value % base_destino

        if remainder in diccionario:

            result = diccionario[remainder] + result

            decimal_value //= base_destino
    return result
```

En esta funcion primero verifica si decimal_value es 0 para retornar 0 en dado caso. si no es 0 inicializa una variable llamada result como un str (se devolvera los datos en simbolos) entra en un while mientras el decimal_value sea mayor a 0 va a asignarle a remainder el residuo de decimal_value y base_destino (se hara una division) si remainder esta dentro del diccionario como una llave entonces asignara a resultado el valor del item del diccionario con la llave remainder mas el resultado esto hara que cada valor nuevo se ponga justo delante de los

anteriores obteniendo un numero en orden. luego se le quita el digito menos significativo a decimal_value. cuando decimal_value no tenga mas numeros retornara el resultado.

```
def rela_potencias_mayor_a_menor(base_destino,lista_elementos,base_origen):
    divisor = base_destino
    base = base_origen
    potencia = encontrar_relacion_potencias(divisor,base)
    tamaño_lista=len(lista_elementos)
    resultado_final=""
    primer_numero=""
    for x in range(tamaño_lista):
        nume_a_operar = int(lista_elementos[tamaño_lista-x-1])
        for exp in range (potencia+1):
            cociente = nume_a_operar // (divisor**(int(potencia-exp)))
            residuo = nume_a_operar % (divisor**(int(potencia-exp)))
            primer_numero = str(primer_numero)+str(cociente)
            nume_a_operar = residuo
        resultado_final=str(int(primer_numero))+resultado_final
        primer_numero=""
    final=int(resultado_final)
    return final
```

En esta funcion asigna el valor a divisor como base_destino y base como base_origen para poder trabajar mas con la logica de la division sin perder el hilo. llama a una funcion encontrar_relacion_potencias enviandole el divisor y la base esto para ver cuantas veces cabe el divisor en la base para tener una idea clara de la potencia. luego se le asigna a tamaño_lista la longitud de la lista_elementos. se inicializan dos variables mas como str resultado_final y primer_numero.

se entra en un ciclo for que se repetira hasta tamaño_lista en este ciclo realizara las siguientes operaciones. transforma la lista en un numero nume_a_operar que se podra operar en x-1 posicion esta suministrada por el for es decir cada que se repita el for cambiara el numero en x-1 posicion de la lista_elementos. luego entrara a un segundo for en este se repetira hasta potencia+1 realizando las siguientes operaciones

asignara a cociente el valor del cociente calculado con el nume_a_operar y el divisor elevado a la potencia-exp esta resta de -exp se hace para que cada iteracion que haga el segundo for baje en un unidad a la potencia. se le asigna a residuo el residuo de nume_a_operar y el divisor elevado a la potencia-exp.

asigna el cociente al final de primer_numero esto hara que el resultado final este el numero ordenado.

y reasigna nume_a_operar ahora nume_a_operar tendra el valor del residuo. cuando finaliza el segundo for es decir la potencia ya haya pasado por 0 asignara a resultado_final el primer_numero + resultado_final esto nuevamente para que el numero obtenido este en orden.

luego reinicializa la variable primer_numero como “” para que cuando se repita el primer for no esten los datos de la iteración pasada. al final vuelve el resultado como un entero y lo retorna.

```
def encontrar_relacion_potencias(base, numero):
    def encontrar_exponente(base, numero):
        exponente = 0
        while base ** exponente != numero:
            exponente += 1
            if base ** exponente > numero:
                return None  # No es una relación de potencias exacta
        return exponente

    exponente = encontrar_exponente(base, numero)
    if exponente is not None:
        return exponente
    else:
        return "No hay una relación de potencias exacta"
```

en la funcion encontrar_relacion_potencias llama a la funcion encontrar_exponente y le envia la base y el numero comprueba si exponente no es None para retornar el exponente en caso contrario de que sea None retornara un str diciendo que No hay relacion de potencias exacta.

en la funcion encontrar_exponente inicializa exponente en 0 y entra en un ciclo while mientras la base elevada al exponente sea diferente al numero va a realizar lo siguiente aumenta en 1 a exponente y verifica si base elevada a exponente es mayor a numero retornara None. cuando finalice el ciclo retornara exponente.

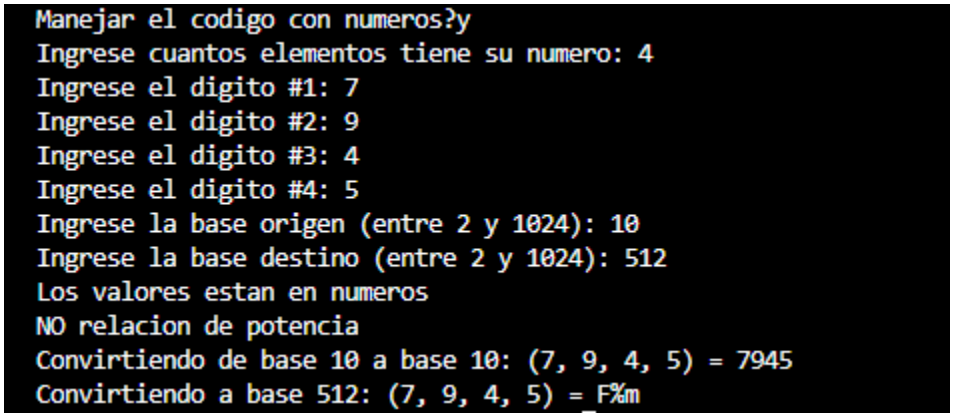
Asi concluiria el codigo cumpliendo con las funciones solicitadas pasar de una base n a una base m verificando si hay relacion de potencias y utilizando diferentes metodos de conversion dependiendo la relacion de potencias. Estos numeros se pueden verificar en base 10 cuando no hay relacion de potencias, para verificar un numero que pase de una base a otra mediante el metodo de potencias seria pasar este mismo numero a base 10 en el codigo para asi obtener el resultado y comparar.

Existe una segunda version del codigo que soporta la busqueda con simbolos es decir en vez de decirle al codigo 9 10 11 12 se le podria decir 9 A B C y entenderia igual ya que tiene implementado un diccionario de datos, sin embargo esta parte aun no soporta el metodo de potencias no lo he adecuado para esto pero si logra hacer conversiones mediante base 10 y devuelve el numero en simbolos.

Resultados

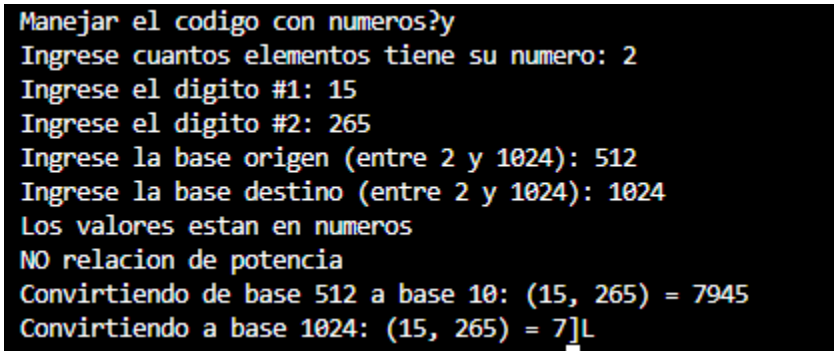
A Continuación mostrare los escenarios de pruebas solicitados durante los avances para comprobar que el codigo funciona perfectamente.

CASO 1: Convertir el numero 7945 que esta en base 10 a base 512. No hay relación de potencia.



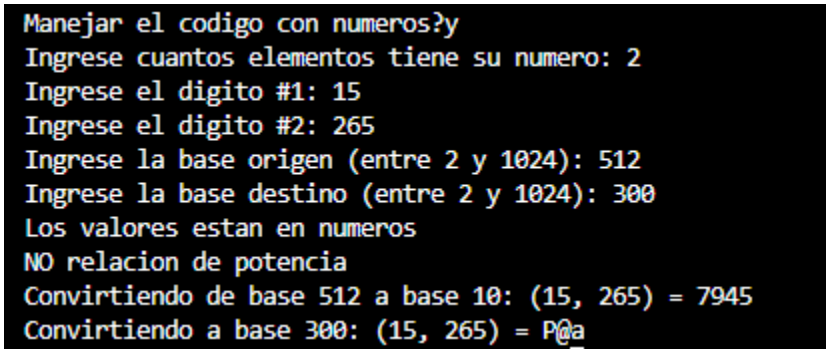
Como podemos observar luego de suministrarle todos los datos pedidos por el codigo este me dice que los valores estan en numeros y no existe una relacion de potencias asi que procede a mostrarme el numero en base 10 y luego en base 512 (este metodo tiene implementado la traduccion de numeros a simbolos) si observamos el diccionario de datos que presente anteriormente podemos verificar que F %m es igual a 15 265 en base 512.

CASO 2: Convertir el numero dado en el caso 1 que esta en base 512 a base 1024. No hay relación de potencia.



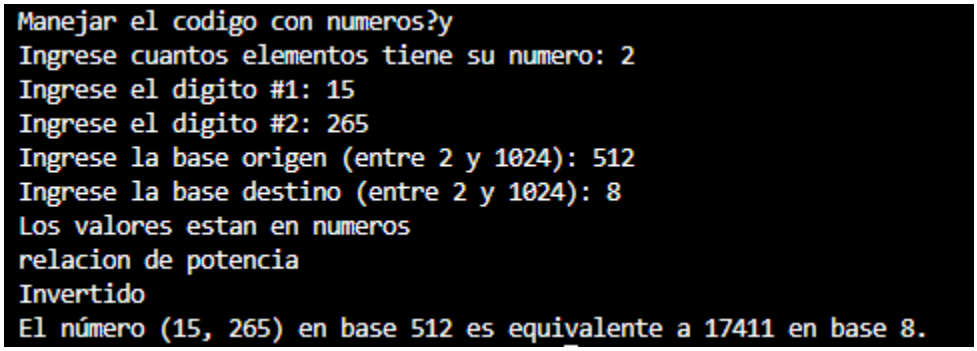
En el caso 2 observamos que no hay relacion de potencia asi que pasa primero por base 10 y nos muestra que 15 265 en base 10 es 7945 que es el numero que primeramente nos habian dado en el caso 1 por lo tanto si es correcto la conversion del caso 1, seguido nos da el resultado del numero en base 1024 que seria 7]L que traducido en numeros seria 7 777 en base 1024.

CASO 3: Convertir el numero dado en el caso 1 que esta en base 512 a base 300. No hay relación de potencia.



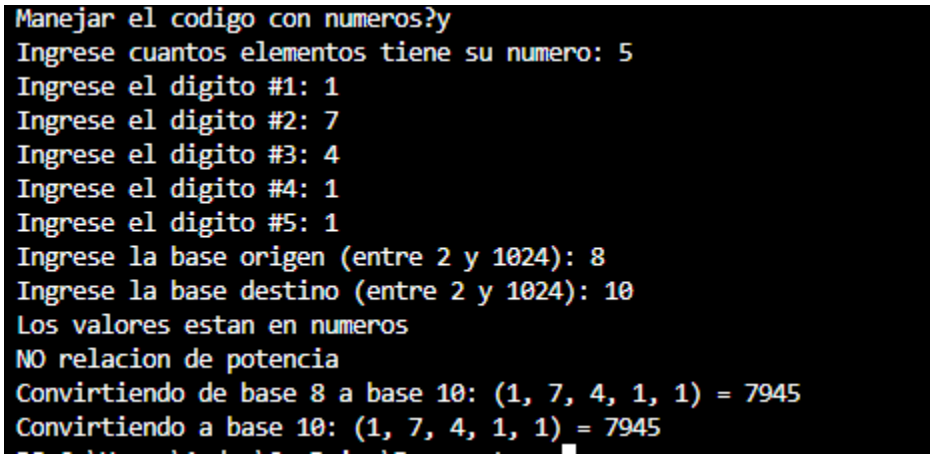
En el caso 3 como no hay relacion de potencia pasa por base 10 primero volviendo a mostrar el numero inicial en base 10 que coincide con el original. seguido pasa a base 300 como resultado da P @a en base 300 que traducido a numeros es 26 145 en base 300

CASO 4: Convertir el numero dado en el caso 1 que esta en abse 512 a base 8. HAY relacion de potencias.



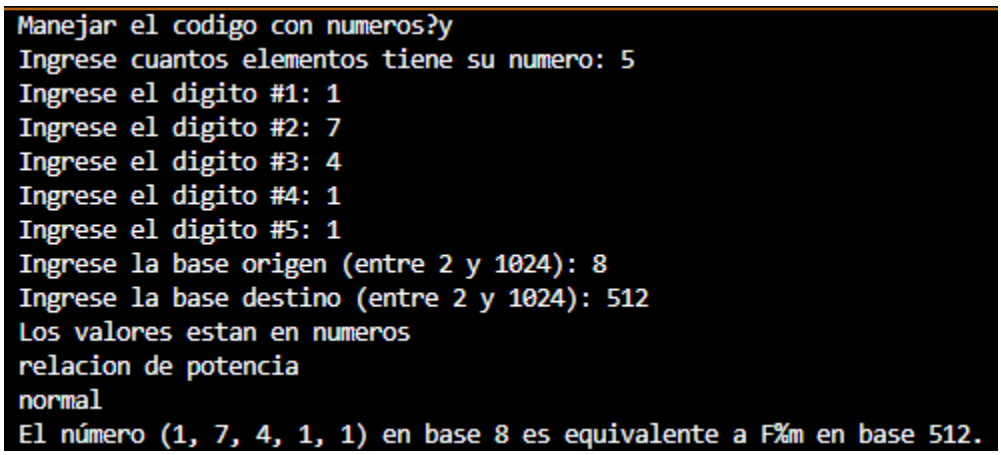
En el caso 4 vemos que hay relacion de potencia pero tambien nos indica que esta invertido con esto queria decir que la base destino es menor a la base origen por lo tanto se aplica un metodo diferente. al ser relacion de potencia pasa directamente a base destino sin tener que pasar por base 10 como resultado nos da 17411 en base 8.

CASO 5: Convertir el numero dado en el caso 4 que esta en base 8 a base 10 para verificar si la conversion fue exitosa con el metodo de potencias.



Al pasarlo a base 10 no hay relacion de potencias y el resultado en base 10 es 7945 coincidiendo con el numero original dado.

CASO BONUS: Convertir el numero dado en el caso 4 a base 512. Hay relacion de potencias se prueba el otro metodo.



Como se puede observar en el caso bonus se comprobo el otro metodo de relacion de potencia. cuando la base destino es mayor a la base origen. el resultado de esta conversion es F %m que es el mismo resultado del caso 1 por lo tanto podemos afirmar que las conversiones hechos has sido exitosas.

Conclusión:

La conversión entre bases numéricas es una tarea que presenta desafíos significativos debido a las diversas estrategias disponibles para realizar dichas conversiones. A lo largo de este laboratorio, se exploraron diferentes métodos y enfoques con el propósito de diseñar un sistema de conversión numérica versátil, eficaz y preciso. El objetivo principal era desarrollar un sistema capaz de operar en un amplio rango de bases, desde 2 hasta 1024, y de cumplir con condiciones específicas que aseguren la exactitud de los resultados. El diseño y la implementación del sistema exigieron una comprensión profunda de las propiedades y relaciones entre las bases numéricas, así como la capacidad de adaptarse a situaciones en las que estas bases podrían tener o no relaciones de potencias. Aunque queda margen para mejoras en el código, especialmente en la parte que maneja los símbolos y las operaciones, el sistema logró cumplir con **éxito** el propósito fundamental del laboratorio: presentar una solución funcional para la conversión de bases numéricas bajo condiciones específicas. El aprendizaje y estudio de los métodos existentes, aunque no todos fueran de autoría propia, contribuyeron a la creación de un sistema sólido y funcional. Este laboratorio reforzó la idea de que el diseño lógico es una disciplina en constante evolución, donde la exploración de enfoques existentes y la adaptación creativa son esenciales para lograr soluciones eficientes y confiables

En conclusión, este laboratorio ha sido una valiosa oportunidad para aplicar los conocimientos teóricos en un contexto práctico, además de brindar un espacio para el desarrollo y la implementación de soluciones ingeniosas. Los resultados obtenidos y los desafíos enfrentados aportan una base sólida para futuras iteraciones y mejoras en el sistema de conversión multibase.

Referencias:

w3schools. 2023. Python Tutorial. https://www.w3schools.com/python/

Parzibyte. 2020. Conversor de números en Python. https://parzibyte.me/blog/2020/12/15/conversor-numeros-python/

Profeacademic. 2017. NUMERACIÓN: CONVERTIR DE UNA BASE A OTRA (CASO ESPECIAL) https://www.youtube.com/watch?v=M6f2x5mmnwQ&ab_channel=Profeacademic