

Some Revision Exercises

June 6, 2023

- Let n and p be two positive integers. The multiplicity of p as a factor of n is the maximum integer exponent m such that

$$n = p^m \times q \rightarrow \text{factor of } n$$

factor of n (pointing to p^m) $m = \max.$ $q \rightarrow \text{factor of } n$

for some integer q . Particularly, q is a factor of n that can't be divided evenly by p . If p is not a factor of n , then the multiplicity of p as a factor of n is defined to be 0. As examples, shown in Table 1 are the multiplicities of some numbers as factors of 137200.

Write a Python function with the `def` statement

$m \leq 0$

$p_1, p_2, p_3, p_4, p_5, \dots$

For in p

while $n \% p_1 \neq 0$

$n = n // p_1$

$m += 1$

multiplicity: m

Factor	Multiplicity	Factorisation of 137200
2	4	$2^4 \times 8575$
3	0	$3^0 \times 137200$
5	2	$5^2 \times 5488$
7	3	$7^3 \times 400$
70	2	$70^2 \times 28$
2744	1	$2744^1 \times 50$

137200

$2 \cdot 2 \cdot 2 \cdot 2$

is a factor of n that can be divided evenly by p^m .

Table 1: Some factors of 137200 and their multiplicities as factors of the 137200.

```
def count_multiplicity(n, *p)
```

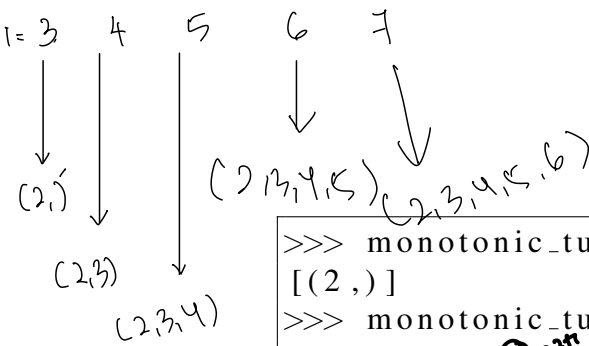
that returns the multiplicity of each element of p as a factor of the positive integer n in a dictionary whose keys are the elements of p with associated values being the respective multiplicities. For instance, the following sample call generates the results as shown in Table 1:

```
>>> count_multiplicity(137200, 2, 3, 5, 7, 70, 2744)
{2: 4, 3: 0, 5: 2, 7: 3, 70: 2, 2744: 1}
```

137200 [2, 3, 5, 7, 70, 2744]

mult \rightarrow $\{0:0, 3:0, 5:0, 7:0, 70:0, 2744:0\}$.

1 2 3 4 5 6 7 8 9 10 11
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 473, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 687, 691, 697, 701, 709, 713, 719, 727, 733, 739, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 833, 839, 847, 853, 857, 859, 863, 877, 881, 883, 887, 893, 899, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 973, 977, 983, 989, 991, 993, 997, 1000, 1001, 1003, 1007, 1009, 1013, 1017, 1019, 1021, 1023, 1027, 1031, 1033, 1037, 1039, 1043, 1047, 1049, 1051, 1053, 1057, 1059, 1063, 1067, 1069, 1073, 1077, 1081, 1087, 1091, 1093, 1097, 1103, 1107, 1109, 1113, 1117, 1121, 1123, 1127, 1129, 1133, 1137, 1139, 1143, 1147, 1149, 1153, 1157, 1159, 1163, 1167, 1169, 1173, 1177, 1181, 1183, 1187, 1193, 1197, 1201, 1203, 1207, 1211, 1213, 1217, 1219, 1223, 1227, 1229, 1231, 1233, 1237, 1239, 1243, 1247, 1249, 1253, 1257, 1259, 1263, 1267, 1269, 1273, 1277, 1279, 1283, 1287, 1289, 1293, 1297, 1301, 1303, 1307, 1309, 1313, 1317, 1319, 1323, 1327, 1329, 1333, 1337, 1339, 1343, 1347, 1349, 1353, 1357, 1359, 1363, 1367, 1369, 1373, 1377, 1379, 1383, 1387, 1389, 1393, 1397, 1401, 1403, 1407, 1409, 1413, 1417, 1419, 1423, 1427, 1429, 1433, 1437, 1439, 1443, 1447, 1449, 1453, 1457, 1459, 1463, 1467, 1469, 1473, 1477, 1479, 1483, 1487, 1489, 1493, 1497, 1501, 1503, 1507, 1509, 1513, 1517, 1519, 1523, 1527, 1529, 1531, 1533, 1537, 1539, 1543, 1547, 1549, 1553, 1557, 1559, 1563, 1567, 1569, 1573, 1577, 1579, 1583, 1587, 1589, 1593, 1597, 1601, 1603, 1607, 1609, 1613, 1617, 1619, 1623, 1627, 1629, 1633, 1637, 1639, 1643, 1647, 1649, 1653, 1657, 1659, 1663, 1667, 1669, 1673, 1677, 1679, 1683, 1687, 1689, 1693, 1697, 1701, 1703, 1707, 1709, 1713, 1717, 1719, 1723, 1727, 1729, 1733, 1737, 1739, 1743, 1747, 1749, 1753, 1757, 1759, 1763, 1767, 1769, 1773, 1777, 1779, 1783, 1787, 1789, 1793, 1797, 1801, 1803, 1807, 1809, 1813, 1817, 1819, 1823, 1827, 1829, 1831, 1833, 1837, 1839, 1843, 1847, 1849, 1853, 1857, 1859, 1863, 1867, 1869, 1873, 1877, 1879, 1883, 1887, 1889, 1893, 1897, 1901, 1903, 1907, 1909, 1913, 1917, 1919, 1923, 1927, 1929, 1931, 1933, 1937, 1939, 1943, 1947, 1949, 1953, 1957, 1959, 1963, 1967, 1969, 1973, 1977, 1979, 1983, 1987, 1989, 1993, 1997, 2001, 2003, 2007, 2009, 2013, 2017, 2019, 2023, 2027, 2029, 2031, 2033, 2037, 2039, 2043, 2047, 2049, 2053, 2057, 2059, 2063, 2067, 2069, 2073, 2077, 2079, 2083, 2087, 2089, 2093, 2097, 2101, 2103, 2107, 2109, 2113, 2117, 2119, 2123, 2127, 2129, 2131, 2133, 2137, 2139, 2143, 2147, 2149, 2153, 2157, 2159, 2163, 2167, 2169, 2173, 2177, 2179, 2183, 2187, 2189, 2193, 2197, 2201, 2203, 2207, 2209, 2213, 2217, 2219, 2223, 2227, 2229, 2231, 2233, 2237, 2239, 2243, 2247, 2249, 2253, 2257, 2259, 2263, 2267, 2269, 2273, 2277, 2279, 2283, 2287, 2289, 2293, 2297, 2301, 2303, 2307, 2309, 2313, 2317, 2319, 2323, 2327, 2329, 2331, 2333, 2337, 2339, 2343, 2347, 2349, 2353, 2357, 2359, 2363, 2367, 2369, 2373, 2377, 2379, 2383, 2387, 2389, 2393, 2397, 2401, 2403, 2407, 2409, 2413, 2417, 2419, 2423, 2427, 2429, 2431, 2433, 2437, 2439, 2443, 2447, 2449, 2453, 2457, 2459, 2463, 2467, 2469, 2473, 2477, 2479, 2483, 2487, 2489, 2493, 2497, 2501, 2503, 2507, 2509, 2513, 2517, 2519, 2523, 2527, 2529, 2531, 2533, 2537, 2539, 2543, 2547, 2549, 2553, 2557, 2559, 2563, 2567, 2569, 2573, 2577, 2579, 2583, 2587, 2589, 2593, 2597, 2601, 2603, 2607, 2609, 2613, 2617, 2619, 2623, 2627, 2629, 2631, 2633, 2637, 2639, 2643, 2647, 2649, 2653, 2657, 2659, 2663, 2667, 2669, 2673, 2677, 2679, 2683, 2687, 2689, 2693, 2697, 2701, 2703, 2707, 2709, 2713, 2717, 2719, 2723, 2727, 2729, 2731, 2733, 2737, 2739, 2743, 2747, 2749, 2753, 2757, 2759, 2763, 2767, 2769, 2773, 2777, 2779, 2783, 2787, 2789, 2793, 2797, 2801, 2803, 2807, 2809, 2813, 2817, 2819, 2823, 2827, 2829, 2831, 2833, 2837, 2839, 2843, 2847, 2849, 2853, 2857, 2859, 2863, 2867, 2869, 2873, 2877, 2879, 2883, 2887, 2889, 2893, 2897, 2901, 2903, 2907, 2909, 2913, 2917, 2919, 2923, 2927, 2929, 2931, 2933, 2937, 2939, 2943, 2947, 2949, 2953, 2957, 2959, 2963, 2967, 2969, 2973, 2977, 2979, 2983, 2987, 2989, 2993, 2997, 3001, 3003, 3007, 3009, 3013, 3017, 3019, 3023, 3027, 3029, 3031, 3033, 3037, 3039, 3043, 3047, 3049, 3053, 3057, 3059, 3063, 3067, 3069, 3073, 3077, 3079, 3083, 3087, 3089, 3093, 3097, 3101, 3103, 3107, 3109, 3113, 3117, 3119, 3123, 3127, 3129, 3131, 3133, 3137, 3139, 3143, 3147, 3149, 3153, 3157, 3159, 3163, 3167, 3169, 3173, 3177, 3179, 3183, 3187, 3189, 3193, 3197, 3201, 3203, 3207, 3209, 3213, 3217, 3219, 3223, 3227, 3229, 3231, 3233, 3237, 3239, 3243, 3247, 3249, 3253, 3257, 3259, 3263, 3267, 3269, 3273, 3277, 3279, 3283, 3287, 3289, 3293, 3297, 3301, 3303, 3307, 3309, 3313, 3317, 3319, 3323, 3327, 3329, 3331, 3333, 3337, 3339, 3343, 3347, 3349, 3353, 3357, 3359, 3363, 3367, 3369, 3373, 3377, 3379, 3383, 3387, 3389, 3393, 3397, 3401, 3403, 3407, 3409, 3413, 3417, 3419, 3423, 3427, 3429, 3431, 3433, 3437, 3439, 3443, 3447, 3449, 3453, 3457, 3459, 3463, 3467, 3469, 3473, 3477, 3479, 3483, 3487, 3489, 3493, 3497, 3501, 3503, 3507, 3509, 3513, 3517, 3519, 3523, 3527, 3529, 3531, 3533, 3537, 3539, 3543, 3547, 3549, 3553, 3557, 3559, 3563, 3567, 3569, 3573, 3577, 3579, 3583, 3587, 3589, 3593, 3597, 3601, 3603, 3607, 3609, 3613, 3617, 3619, 3623, 3627, 3629, 3631, 3633, 3637, 3639, 3643, 3647, 3649, 3653, 3657, 3659, 3663, 3667, 3669, 3673, 3677, 3679, 3683, 3687, 3689, 3693, 3697, 3701, 3703, 3707, 3709, 3713, 3717, 3719, 3723, 3727, 3729, 3731, 3733, 3737, 3739, 3743, 3747, 3749, 3753, 3757, 3759, 3763, 3767, 3769, 3773, 3777, 3779, 3783, 3787, 3789, 3793, 3797, 3801, 3803, 3807, 3809, 3813, 3817, 3819, 3823, 3827, 3829, 3831, 3833, 3837, 3839, 3843, 3847, 3849, 3853, 3857, 3859, 3863, 3867, 3869, 3873, 3877, 3879, 3883, 3887, 3889, 3893, 3897, 3901, 3903, 3907, 3909, 3913, 3917, 3919, 3923, 3927, 3929, 3931, 3933, 3937, 3939, 3943, 3947, 3949, 3953, 3957, 3959, 3963, 3967, 3969, 3973, 3977, 3979, 3983, 3987, 3989, 3993, 3997, 4001, 4003, 4007, 4009, 4013, 4017, 4019, 4023, 4027, 4029, 4031, 4033, 4037, 4039, 4043, 4047, 4049, 4053, 4057, 4059, 4063, 4067, 4069, 4073, 4077, 4079, 4083, 4087, 4089, 4093, 4097, 4101, 4103, 4107, 4109, 4113, 4117, 4119, 4123, 4127, 4129, 4131, 4133, 4137, 4139, 4143, 4147, 4149, 4153, 4157, 4159, 4163, 4167, 4169, 4173, 4177, 4179, 4183, 4187, 4189, 4193, 4197, 4201, 4203, 4207, 4209, 4213, 4217, 4219, 4223, 4227, 4229, 4231, 4233, 4237, 4239, 4243, 4247, 4249, 4253, 4257, 4259, 4263, 4267, 4269, 4273, 4277, 4279, 4283, 4287, 4289, 4293, 4297, 4301, 4303, 4307, 4309, 4313, 4317, 4319, 4323, 4327, 4329, 4331, 4333, 4337, 4339, 4343, 4347, 4349, 4353, 4357, 4359, 4363, 4367, 4369, 4373, 4377, 4379, 4383, 4387, 4389, 4393, 4397, 4401, 4403, 4407, 4409, 4413, 4417, 4419, 4423, 4427, 4429, 4431, 4433, 4437, 4439, 4443, 4447, 4449, 4453, 4457, 4459, 4463, 4467, 4469, 4473, 4477, 4479, 4483, 4487, 4489, 4493, 4497, 4501, 4503, 4507, 4509, 4513, 4517, 4519, 4523, 4527, 4529, 4531, 4533, 4537, 4539, 4543, 4547, 4549, 4553, 4557, 4559, 4563, 4567, 4569, 4573, 4577, 4579, 4583, 4587, 4589, 4593, 4597, 4601, 4603, 4607, 4609, 4613, 4617, 4619, 4623, 4627, 4629, 4631, 4633, 4637, 4639, 4643, 4647, 4649, 4653, 4657, 4659, 4663, 4667, 4669, 4673, 4677, 4679, 4683, 4687, 4689, 4693, 4697, 4701, 4703, 4707, 4709, 4713, 4717, 4719, 4723, 4727, 4729, 4731, 4733, 4737, 4739, 4743, 4747, 4749, 4753, 4757, 4759, 4763, 4767, 4769, 4773, 4777, 4779, 4783, 4787, 4789, 4793, 4797, 4801, 4803, 4807, 4809, 4813, 4817, 4819, 4823, 4827, 4829, 4831, 4833, 4837, 4839, 4843, 4847, 4849, 4853, 4857, 4859, 4863, 4867, 4869, 4873, 4877, 4879, 4883, 4887, 4889, 4893, 4897, 4901, 4903, 4907, 4909, 4913, 4917, 4919, 4923, 4927, 4929, 4931, 4933, 4937, 4939, 4943, 4947, 4949, 4953, 4957, 4959, 4963, 4967, 4969, 4973, 4977, 4979, 4983, 4987, 4989, 4993, 4997, 5001, 5003, 5007, 5009, 5013, 5017, 5019, 5023, 5027, 5029, 5031, 5033, 5037, 5039, 5043, 5047, 5049, 5053, 5057, 5059, 5063, 5067, 5069, 5073, 5077, 5079, 5083, 5087, 5089, 5093, 5097, 5101, 5103, 5107, 5109, 5113, 5117, 5119, 5123, 5127, 5129, 5131, 5133, 5137, 5139, 5143, 5147, 5149, 5153, 5157, 5159, 5163, 5167, 5169, 5173, 5177, 5179, 5183, 5187, 5189, 5193, 5197, 5201, 5203, 5207, 5209, 5213, 5217, 5219, 5223, 5227, 5229, 5231, 5233, 5237, 5239, 5243, 5247, 5249, 5253, 5257, 5259, 5263, 5267, 5269, 5273, 5277, 5279, 5283, 5287, 5289, 5293, 5297, 5301, 5303, 5307, 5309, 5313, 5317, 5319, 5323, 5327, 5329, 5331, 5333, 5337, 5339, 5343, 5347, 5349, 5353, 5357, 5359, 5363, 5367, 5369, 5373, 5377, 5379, 5383, 5387, 5389, 5393, 5397, 5401, 5403, 5407, 5409, 5413, 5417, 5419, 5423, 5427, 5429, 5431, 5433, 5437, 5439, 5443, 5447, 5449, 5453, 5457, 5459, 5463, 5467, 5469, 5473, 5477, 5479, 5483, 5487, 5489, 5493, 5497, 5501, 5503, 5507, 5509, 5513, 5517, 5519, 5523, 5527, 5529, 5531, 5533, 5537, 5539, 5543, 5547, 5549, 5553, 5557, 5559, 5563, 5567, 5569, 5573, 5577, 5579, 5583, 5587, 5589, 5593, 5597, 5601, 5603, 5607, 5609, 5613, 5617, 5619, 5623, 5627, 5629, 5631, 5633, 5637, 5639, 5643, 5647, 5649, 5653, 5657, 5659, 5663, 5667, 5669, 5673, 5677, 5679, 5683, 5687, 5689, 5693, 5697, 5701, 5703, 5707, 5709, 5713, 5717, 5719, 5723, 5727, 5729, 5731, 5733, 5737, 5739, 5743, 5747, 5749, 5753, 5757, 5759, 5763, 5767, 5769, 5773, 5777, 5779, 5783, 5787, 5789, 5793, 5797, 5801, 5803, 5807, 5809, 5813, 5817, 5819, 5823, 5827, 5829, 5831, 5833, 5837, 5839, 5843, 5847, 5849, 5853, 5857, 5859, 5863, 5867, 5869, 5873, 5877, 5879, 5883, 5887, 5889, 5893, 5897, 5901, 5903, 5907, 5909, 5913, 5917, 5919, 5923, 5927, 5929, 5931, 5933, 5937, 5939, 5943, 5947, 5949, 5953, 5957, 5959, 5963, 5967, 5969, 5973, 5977, 5979, 5983, 5987, 5989, 5993, 5997, 6001, 6003, 6007, 6009, 6013, 6017, 6019, 6023, 6027, 6029, 6031, 6033, 6037, 6039, 6043, 6047, 6049, 6053, 6057, 6059, 6063, 6067, 6069, 6073, 6077, 6079, 6083, 6087, 6089, 6093, 6097, 6101, 6103, 6107, 6109, 6113, 6117, 6119, 6123, 6127, 6129, 6131, 6133, 6137, 6139, 6143, 6147, 6149, 6153, 6157, 6159, 6163, 6167, 6169, 6173, 6177, 6179, 6183, 6187, 6189, 6193, 6197, 6201, 6203, 6207, 6209, 6213, 6217, 6219, 6223, 6227, 6229, 6231, 6233, 6237, 6239, 6243, 6247, 6249, 6253, 6257, 6259, 6263, 6267, 6269, 6273, 6277, 6279, 6283, 6287, 6289, 6293, 6297, 6301, 6303, 6307, 6309, 6313, 6317, 6319, 6323, 6327, 6329, 6331, 6333, 6337, 6339, 6343, 6347, 6349, 6353, 6357, 6359, 6363, 6367, 6369, 6373, 6377, 6379, 6383, 6387, 6389, 6393, 6397, 6401, 6403, 6407, 6409, 6413, 6417, 6419, 6423, 6427, 6429, 6431, 6433, 6437, 6439, 6443, 6447, 6449, 6453, 6457, 6459, 6463, 6467, 6469, 6473, 6477, 6479, 6483, 6487, 6489, 6493, 6497, 6501, 6503, 6507, 6509, 6513, 6517, 6519, 6523, 6527, 6529, 6531, 6533, 6537, 6539, 6543, 6547, 6549, 6553, 6557, 6559, 6563, 6567, 6569, 6573, 6577, 6579, 6583, 6587, 6589, 6593, 6597, 6601, 6603, 6607, 6609, 6613, 6617, 6619, 6623, 6627, 6629, 6631, 6633, 6637, 6639, 6643, 6647, 6649, 6653, 6657, 6659, 6663, 6667, 6669, 6673, 6677, 6679, 6683, 6687, 6689, 6693, 6697, 6701, 6703, 6707, 6709, 6713, 6717, 6719, 6723, 6727, 6729, 6731, 6733, 6737, 6739, 6743, 6747, 6749, 6753, 6757, 6759, 6763, 6767, 6769, 6773, 6777, 6779, 6783, 6787, 6789, 6793, 6797, 6801, 6803, 6807, 6809, 6813, 6817, 6819, 6823, 6827, 6829, 6831, 6833, 6837, 6839, 6843, 6847, 6849, 6853, 6857, 6859, 6863, 6867, 6869, 6873, 6877, 6879, 6883, 6887, 6889, 6893, 6897, 6901, 6903, 6907, 6909, 6913, 6917, 6919, 6923, 6927, 6929, 6931, 6933, 6937, 6939, 6943, 6947, 6949, 6953, 6957, 6959, 6963, 6967, 6969, 6973, 6977, 6979, 6983, 6987, 6989, 6993, 6997, 7001, 7003, 7007, 7009, 7013, 7017, 7019, 7023, 7027, 7029, 7031, 7033, 7037, 7039, 7043, 7047, 7049, 7053, 7057, 7059, 7063, 7067, 7069, 7073, 7077, 7079, 7083, 7087, 7089, 7093,



```
>>> monotonic_tuples(2, 2)
[(2,)]
>>> monotonic_tuples(2, 6)
[(2,), (2, 3), (2, 3, 4), (2, 3, 4, 5), (2, 3, 4, 5, 6)]
>>> monotonic_tuples(6, 2)
[(6, 5, 4, 3, 2), (5, 4, 3, 2), (4, 3, 2), (3, 2), (2,)]
>>> monotonic_tuples(-3, 3)
[(-3,), (-3, -2), (-3, -2, -1), (-3, -2, -1, 0), (-3, ...
-2, -1, 0, 1), (-3, -2, -1, 0, 1, 2), (-3, -2, -1, ...
0, 1, 2, 3)]
>>> monotonic_tuples(3, -3)
[(3, 2, 1, 0, -1, -2, -3), (2, 1, 0, -1, -2, -3), (1, ...
0, -1, -2, -3), (0, -1, -2, -3), (-1, -2, -3), (-2, ...
-3), (-3,)]
```

4. Consider the following list of lists:

```
[[9, -1, 3, 9, -1], [-5, 5, -5, 4, 8], [0, 3, 4, -7, ...
6], [0, 9, -6, -3, 9], [-4, -8, 5, -7, 9], [0, 5, ...
-7, 6, -1]]
```

We say that 8 precedes 6 in this case because 8 first appears in the second list, while 6 first appears in the third list. Likewise, 9 precedes 0 because 9 first appears in the first list while 0 first appears in the third list. If two numbers, such as 9 and -1, first appear in the same list, then we say that they precede each other.

Write a Python function with the `def` statement

```
def precede(lst, a, b)
```

where `lst` is a list of lists and `a` and `b` are integers, that

- returns `None` if either `a` or `b` is not contained in any list in `lst`,
- returns `True` if `a` and `b` are contained in some lists of `lst` and `a` precedes `b`, and
- returns `False` if `a` and `b` are contained in some lists of `lst` but `a` does not precede `b`.

Some sample calls are as follows:

```

>>> lst = [[9, -1, 3, 9, -1], [-5, 5, -5, 4, 8], [0, ...
          3, 4, -7, 6], [0, 9, -6, -3, 9], [-4, -8, 5, -7, 9], ...
          [0, 5, -7, 6, -1]]
>>> precede(lst, 9, -1)
True
>>> precede(lst, 5, -1)
False
>>> precede(lst, 4, 6)
True
>>> precede(lst, 4, 2)
>>> print(precede(lst, 4, 2))
None

```

5. Any function f that maps a range of consecutive integers $\{m, m+1, \dots, n\}$ into itself for some integers m and n , where $m \leq n$, can be encoded as a list. For instance, the function f such that

$$f(-1) = 0, f(0) = 3, f(1) = 4, f(2) = 3, f(3) = 1 \text{ and } f(4) = -1$$

can be encoded as the list

$f = [0, 3, 4, 3, 1, -1]$

with the understanding that $f[0]$ takes the value of $f(-1)$, $f[1]$ takes the value of $f(0)$, and so on. Write a Python function with the `def` statement

```
def composite(f, g, m, n)
```

that returns the list encoding the composite $g \circ f$ of two functions f and g , represented by f and g as lists, sharing the domain from the integer m to the integer n , where $m \leq n$, such that $(g \circ f)(x) = g(f(x))$ for all x in the common domain. For instance,

```

>>> f = [0, 3, 4, 3, 1, -1]
>>> g = [-1, 2, 4, 1, 3, 0]
>>> composite(f, g, -1, 4)
[2, 3, 0, 3, 4, -1]

```

Here f and g are functions such that

$$f(-1) = 0, f(0) = 3, f(1) = 4, f(2) = 3, f(3) = 1, \text{ and } f(4) = -1$$

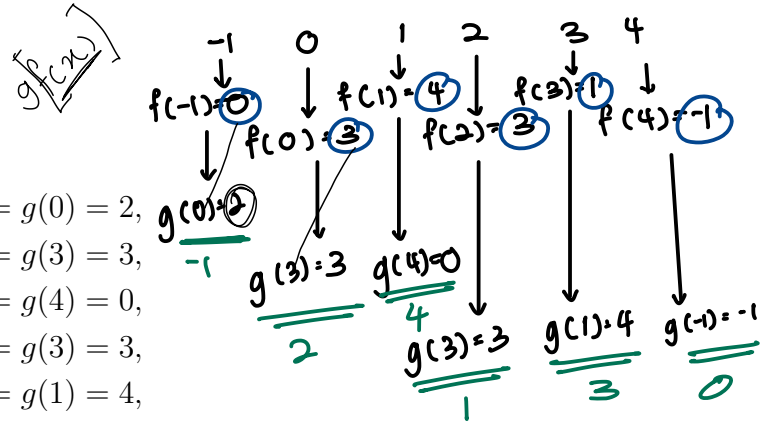
and

$$g(-1) = -1, g(0) = 2, g(1) = 4, g(2) = 1, g(3) = 3, \text{ and } g(4) = 0.$$

0, 3, 4, 3, 1, -1
 (-1, 2, 4, 1, 3, 0)

Hence

$$\begin{aligned} g(f(-1)) &= g(0) = 2, \\ g(f(0)) &= g(3) = 3, \\ g(f(1)) &= g(4) = 0, \\ g(f(2)) &= g(3) = 3, \\ g(f(3)) &= g(1) = 4, \\ g(f(4)) &= g(-1) = -1. \end{aligned}$$



6. Recall that a numerical series $\sum_{k=1}^{\infty} (-1)^k a_k$ is said to be alternating provided $a_k \geq 0$ for all $k \in \mathbb{N}$ or $a_k \leq 0$ for all $k \in \mathbb{N}$. If

(i) $|a_k| \geq |a_{k+1}|$ for all $k \in \mathbb{N}$ and

(ii) $\lim_{k \rightarrow \infty} a_k = 0$,

then the **alternating series test** guarantees that $\sum_{k=1}^{\infty} (-1)^k a_k$ converges to a finite limit L . Furthermore,

$$\left| \sum_{k=1}^n (-1)^k a_k - L \right| \leq |a_{n+1}|$$

for any $n \in \mathbb{N}$.

Write a Python function with the `def` statement

```
def alternating_harmonic_series(bound=0.1)
```

that returns an approximate value \tilde{L} to the limit of the alternating harmonic series

$$\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots$$

such that the $\left| \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} - \tilde{L} \right|$ is strictly bounded by the parameter `bound`. Some sample runs are as follows.

```
>>> alternating_harmonic_series()
0.7456349206349207
>>> alternating_harmonic_series(0.001)
0.6936474305598223
>>> alternating_harmonic_series(0.00001)
0.6931521805849815
```

Do it using different approaches.

- (a) Using a `while` loop.
- (b) Using a `for` loop. You may need to import either `math` or `numpy`.
- (c) Without using any loop. You may need to import either `math` or `numpy`.

7. (Adapted from Exam of Semester 1, Session 2022/2023.) For a sufficiently smooth function f on a bounded interval $[0, b]$, the value of the integral

$$I_b[f] = \int_0^b f(x) dx$$

can be evaluated numerically using the composite trapezoidal rule as follows. Let

$$[x_0, x_1], [x_1, x_2], \dots, [x_{n-1}, x_n]$$

be a partition of the interval $[0, b]$ into subintervals of equal length $h_n = b/n$ such that

$$x_k = kh_n$$

for each $k \in \{0, 1, \dots, n\}$; particularly, $x_0 = 0$ and $x_n = b$. Let

trapezoidal rule.

$$\longrightarrow T_b[f; n] = h_n \left[\frac{1}{2}f(x_0) + f(x_1) + f(x_2) + \dots + f(x_{n-1}) + \frac{1}{2}f(x_n) \right].$$

Then

$$|I_b[f] - T_b(f; n)| \leq \frac{b^3}{12n^2} \max_{x \in [0, b]} |f''(x)| \leq \text{bound} \quad (*)$$

where the term on the right-hand side is an error bound.

In what follows, let

$$f(x) = e^{-x^2}.$$

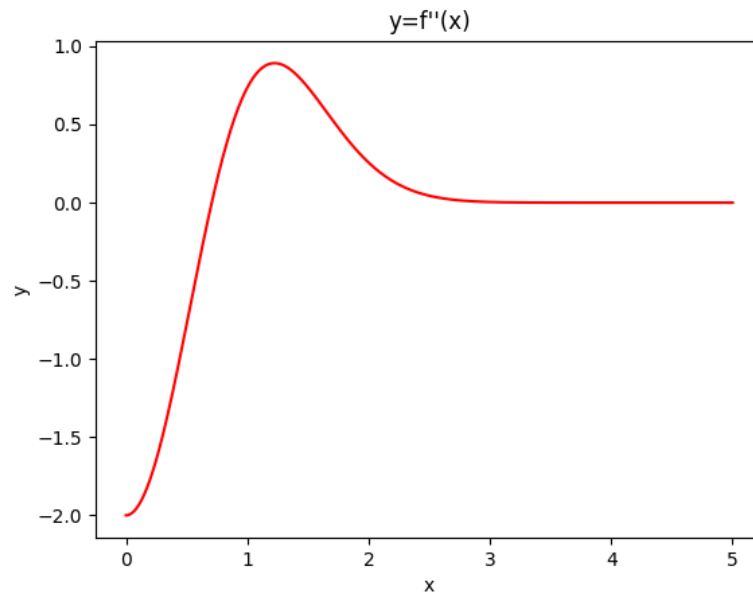
- (a) With the aid of `matplotlib` and `numpy` modules, write a Python script to plot the graph of $f''(x)$ in red on $[0, 5]$ with 5000 evenly spaced points. The plot should look as follows.

matplotlib
numpy

24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0

$h_n = 10/23$

$\frac{\text{limit}^3}{12n^2} \leq \text{bound}$
 $\frac{\text{limit}^3}{12n^2} \leq 12n^2$
 $\sqrt{\frac{\text{limit}^3}{12 \text{ bound}}} \leq n$



Note that $\max_{[0,b]} |f''(x)|$ is attained at $x = 0$. You may use your knowledge of calculus to verify this.

- (b) With the aid of only the **numpy** module, write a Python function with the `def` statement

```
def T(limit , bound)
```

that returns the value of $T_b[f;n]$, where `limit` denotes the value of the limit of integration b , while the error bound stipulated in (*) should be no more than the parameter `bound`.

Let us look at some sample function calls.

```
>>> T(1 , 0.1)
0.731370251828563
>>> T(1 , 0.01)
0.744368339763667
>>> T(1 , 0.001)
0.7464612610366896
>>> T(1 , 0.0001)
0.7467876578237479
```

The exact value of $I_1[f]$ is approximately 0.7468241328124279.

Likewise,


```
>>> T(2, 0.1)
0.8806186341245393
>>> T(2, 0.01)
0.8819125868282965
>>> T(2, 0.001)
0.882063560989435
>>> T(2, 0.0001)
0.8820795759852266
```

The exact value of $I_2[f]$ is approximately 0.882081390762423.

As additional requirements, do not use

- any loop and
- the `trapz()` function in **numpy**

in the definition of `T()`.

8. Recall that the probability density function (PDF) of a random variable X with normal distribution $N(\mu, \sigma^2)$ of mean μ and variance σ^2 is

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2},$$

while the cumulative distribution function (CDF) is

$$\begin{aligned} F_X(x) &= \int_{-\infty}^x f_X(t) dt \\ &= \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^x e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2} dt. \end{aligned}$$

- (a) With the aid of **numpy** and the `quad()` function from **scipy.integrate**, write a Python function with the `def` statement

```
def F(x, mean=0, var=1)
```

that returns the values of the CDF F_X evaluated at the **numpy** array `x` elementwise. The parameter `mean` is the mean μ and the parameter `var` is the variance σ^2 of the distribution. Also, you may assume that the PDF f_X has already been defined as follows.

```
# The PDF of a normal distribution with mean 'mean' ...
    and variance 'var'
```

```
def f(x, mean=0, var=1):
    return 1/np.sqrt(2*np.pi*var) * ...
        np.exp(-0.5*(np.array(x)-mean)**2/var)
```

Some sample calls of $F()$ is given in the following.

```
>>> F(-1, 0, 1)
array(0.15865525)
>>> F(0, 0, 1)
array(0.5)
>>> F(2, 0, 1)
array(0.97724987)
>>> x = np.arange(-3, 3, 0.5).reshape(3, 4)
>>> x
array([[ -3. , -2.5, -2. , -1.5],
       [ -1. , -0.5,  0. ,  0.5],
       [  1. ,  1.5,  2. ,  2.5]])
>>> F(x, 0, 1)
array([[0.0013499 , 0.00620967, 0.02275013, ...
        0.0668072 ],
       [0.15865525, 0.30853754, 0.5 , ...
        0.69146246],
       [0.84134475, 0.9331928 , 0.97724987, ...
        0.99379033]])
>>> F(x, 2, 3)
array([[0.00194621, 0.00468738, 0.01046067, ...
        0.02165407],
       [0.04163226, 0.07445734, 0.12410654, ...
        0.19323812],
       [0.28185143, 0.386415 , 0.5 , ...
        0.613585 ]])
```

- (b) Note that for every probability $p \in [0, 1]$, there is a unique x -value in $(-\infty, \infty)$ such that

$$F_X(x) - p = 0$$

$$F_X(x) = p$$

$$F(x) - p = 0$$

for any given mean μ and variance σ^2 . So it makes sense to talk about the inverse of F given by

$$F_X^{-1}(p) = x$$

$$F(x) - 0.5 = 0 \quad F(x) = 0.5$$

for all $p \in [0, 1]$. With the aid of **numpy** and the `bisect()` function from **scipy.optimize**, write a Python function with the `def` statement

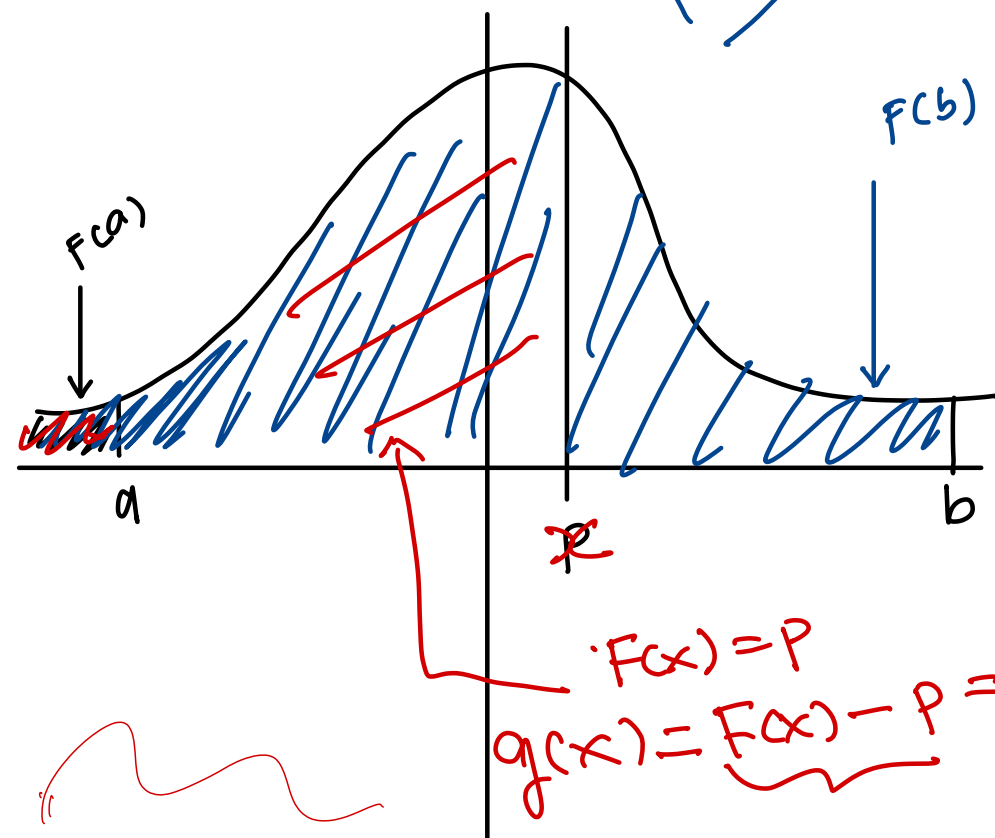
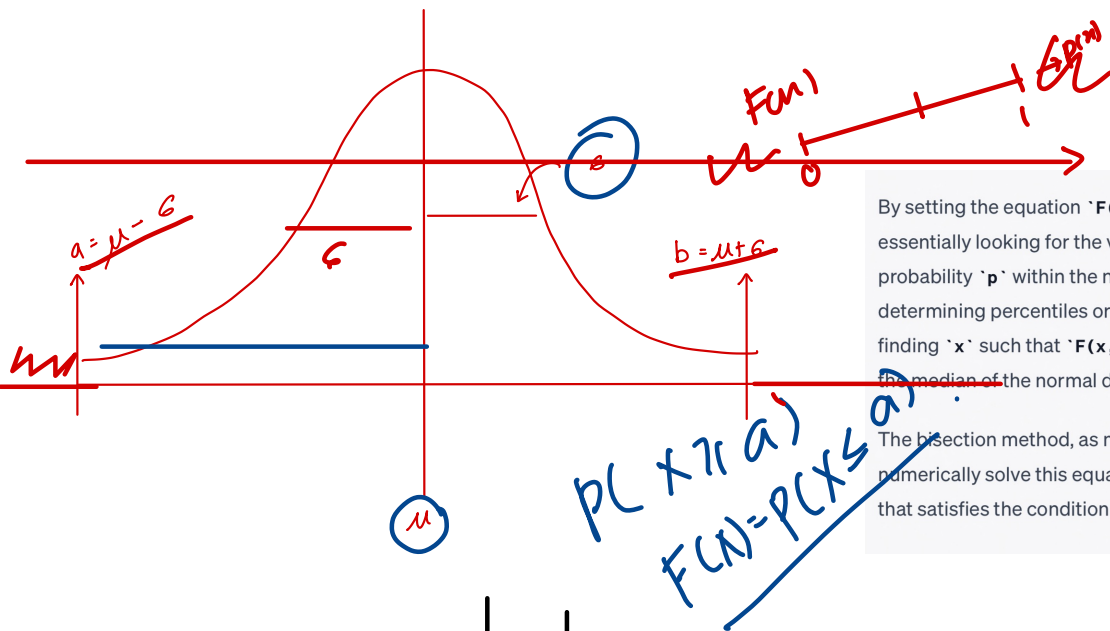
```
def inv_F(prob , mean=0, var=1)
```

that returns the values of the inverse of F_X evaluated at the **numpy** array `prob` elementwise. As before, the mean μ and the variance σ^2 of the distribution are given by the parameters `mean` and `var`, respectively. Also, if an element of `prob` is out of range, i.e. not within the interval $[0, 1]$, then the corresponding value of the array returned should be `numpy.nan` that denotes that the quantity “**Not a Number**”.

Some sample calls of `inv_F()` are given in the following.

```
>>> inv_F(0.35)
array(-0.38532047)
>>> F(inv_F(0.35))
array(0.35)
>>> inv_F(0.35, 2, 1)
array(1.61467953)
>>> F(inv_F(0.35, 2, 1), 2, 1)
array(0.35)
>>> prob = np.arange(0, 1, 0.1).reshape(2,5)
>>> prob
array([[0. , 0.1, 0.2, 0.3, 0.4],
       [0.5, 0.6, 0.7, 0.8, 0.9]])
>>> inv_F(prob)
array([[ -3.90000000e+01, -1.28155157e+00, ...,
        -8.41621205e-01,
        -5.24400513e-01, -2.53347103e-01],
       [ 1.81898940e-12,  2.53347103e-01, ...,
        5.24400513e-01,
        8.41621234e-01,  1.28155157e+00]])
>>> F(inv_F(prob))
array([[0. , 0.1, 0.2, 0.3, 0.4],
       [0.5, 0.6, 0.7, 0.8, 0.9]])
>>> inv_F(prob, 2, 1)
array([[ -37.          ,  0.71844843,  1.15837879, ...,
        1.47559949,
        1.7466529 ],
       [  2.          ,  2.2533471 ,  2.52440051, ...,
        2.84162123,
        3.28155157]])
>>> F(inv_F(prob, 2, 1), 2, 1)
array([[0. , 0.1, 0.2, 0.3, 0.4],
       [0.5, 0.6, 0.7, 0.8, 0.9]])
```

```
>>> prob = np.array([0.4, -0.5, 0.7, 1.4])
>>> prob
array([ 0.4, -0.5, 0.7, 1.4])
>>> inv_F(prob)
array([-0.2533471,          nan,  0.52440051, ...
        nan])
```



lambda $\mu = 4 - \mu$

```
>>> prob = np.array([0.4, -0.5, 0.7, 1.4])
>>> prob
array([ 0.4, -0.5, 0.7, 1.4])
>>> inv_F(prob)
array([-0.2533471, nan, 0.52440051, ...
       nan])
```

9. Consider the function

$$\text{pulse}(x; a, b, f) = \begin{cases} \cos(x) \sin(f \cdot x) & \text{if } a < x < b; \\ \cos(x) & \text{otherwise} \end{cases}$$

where a , b , and f are parameters with $a < b$ and $f > 0$.

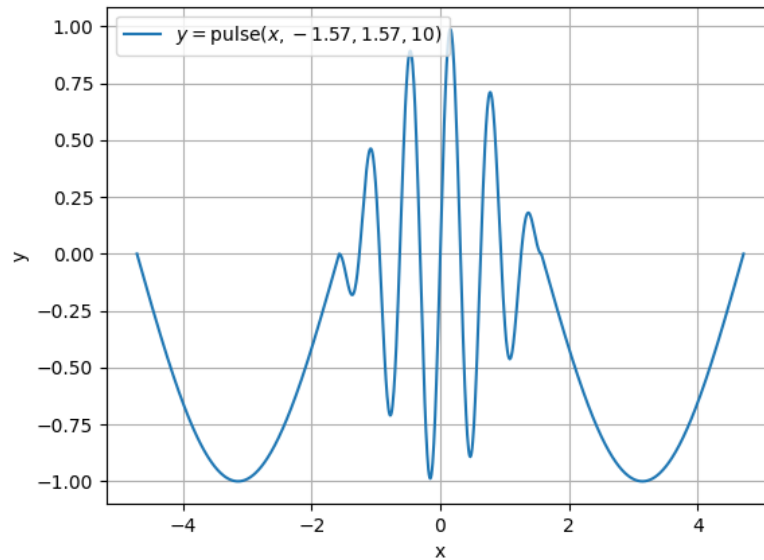
- (a) Using the aid of the **NumPy** module, write a Python function with the `def` statement

```
def pulse(x, a, b, f)
```

that returns the value of the function `pulse` at each element of the **NumPy** array given the parameters `a`, `b`, and `f` that represents the values of a , b , and f , respectively. A sample call of the function is given in the following.

```
>>> x = np.linspace(-np.pi, np.pi, 10)
>>> x.reshape(2, 5)
array([[ -3.14159265, -2.44346095, -1.74532925, ...
        -1.04719755, -0.34906585],
       [ 0.34906585, 1.04719755, 1.74532925, ...
        2.44346095, 3.14159265]])
>>> pulse(x, -np.pi/2, np.pi/2, 5)
array([-1.          , -0.76604444, -0.17364818, ...
        0.4330127  , -0.92541658,
        0.92541658, -0.4330127  , -0.17364818, -0.76604444, ...
        -1.          ])
```

- (b) With the aid of the **Matplotlib** module and the `pulse()` function you defined in (a), write a Python script to plot the graph of $\text{pulse}(x; -\pi/2, \pi/2, 10)$ as x varies in $[-3\pi/2, 3\pi/2]$ using 1000 evenly spaced points. The graph should look as follows.



- (c) * With the aid of the **Matplotlib** module and the `pulse()` function you defined in (a), write a Python script that shows an animation of the graph of $\text{pulse}(x; -\pi/2, \pi/2, f)$ for $x \in [-\pi, \pi]$ using 1000 evenly space points as f varies from $f = 1$ to $f = 100$ and then back to $f = 1$ with a stepsize of 1. The animation should look like the one shown at https://youtu.be/eoL49F_GcFA.

*This is not for revision purpose.