Started on	Thursday, 23 October 2025, 9:02 AM
State	Finished
Completed on	Thursday, 23 October 2025, 11:41 AM
Time taken	2 hours 38 mins
Marks	100.00/100.00
Grade	**10.00** out of 10.00 (**100**%)

Question 1

Correct

Mark 6.67 out of 6.67

Given the radius of a sphere, calculate and **return** its surface area. Surface area is given by the following:

`A = 4*PI*(r**2)`

where `PI` is the constant `3.14159` and `r` is the `radius` of the sphere.

For example:

Test	Result
`print(find_area(5))`	`314.159`

Answer: (penalty regime: 0 %)

Reset answer

```
def find_area(radius):
    pi = 3.14159
    area = 4*pi* (radius**2)
    return float(area)
```

	Test	Expected	Got	
✓	`print(find_area(5))`	`314.159`	`314.159`	✓

Passed all tests! ✓

Correct

Marks for this submission: 6.67/6.67.

Question 2

Correct

Mark 6.67 out of 6.67

In the function below you are given an IPv4 address as a string in dotted decimal notation, **return** `True` (as a boolean data type) if the address is multicast, otherwise return `False` (as a boolean data type).

IPv4 multicast addresses are those in the range 224.0.0.0 to 239.255.255.255.

NOTE: Do NOT use any imported libraries to complete this function

NOTE: `True` and `False` should be Boolean data types, NOT Strings. Test cases will confirm expected data type is accurate

For example:

Test	Result
multicast_check('223.0.0.0')	False
multicast_check('225.0.0.0')	True

HINT: You only need to check the first octet (224 through 239). The rest of the octets do not matter for the purpose of this challenge.

For example:

Test	Result
`print(multicast_check('223.0.0.0'), type(multicast_check('223.0.0.0'))==bool)`	`False True`

Answer: (penalty regime: 0 %)

Reset answer

```
def multicast_check(addr):
    x = int(addr.split('.')[0])
    if x in range(224,240):
        return True
    else:
        return False
```

	Test	Expected	Got	
✓	`print(multicast_check('223.0.0.0'), type(multicast_check('223.0.0.0'))==bool)`	`False True`	`False True`	✓
✓	`print(multicast_check('225.0.0.0'), type(multicast_check('223.0.0.0'))==bool)`	`True True`	`True True`	✓

Passed all tests! ✓

Correct

Marks for this submission: 6.67/6.67.

Question 3

Correct

Mark 6.67 out of 6.67

Complete the function to **return** the well-known ports as a **list** of **integers**.

Ports 0 through 1023 are considered well-known.

NOTE: This function takes NO parameters, therefore you ARE allowed to hard-code your returned data

For example:

Test	Result
`print(well_known())`	`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 1023]`

Answer: (penalty regime: 0 %)

Reset answer

```
def well_known():
    x = range(0,1024)
    y = list(x)
    return y
    pass

```

	Test	Expected	Got	

	Test	Expected	Got	
✓	`print(well_known())`	`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613,`	`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549,`	✓

	Test	Expected	Got	
		614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 1023]	550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802, 803, 804, 805, 806, 807, 808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818, 819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839, 840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859, 860, 861, 862, 863, 864, 865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884, 885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900, 901, 902, 903, 904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921, 922, 923, 924, 925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945, 946, 947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966, 967, 968, 969, 970, 971, 972, 973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992, 993, 994, 995, 996, 997, 998, 999, 1000, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1009, 1010, 1011, 1012, 1013, 1014, 1015, 1016, 1017, 1018, 1019, 1020, 1021, 1022, 1023]	

Passed all tests! ✓

Correct

Marks for this submission: 6.67/6.67.

Question 4

Correct

Mark 6.67 out of 6.67

Given a string for a `number` spelled out as a word, **return** the number as an integer. The strings passed by `number` will only be 'zero', 'one', 'two', 'three', or 'four'.

For example:

Test	Result
`print(STRtoINT('zero'))`	`0`
`print(STRtoINT('one'))`	`1`

Answer: (penalty regime: 0 %)

Reset answer

```
def STRtoINT(number):
    if number == 'zero':
        return 0

    elif number == 'one':
        return 1


    elif number == 'two':
        return 2

    elif number == 'three':
        return 3

    elif number == 'four':
        return 4

    return int(number)


pass

```

	Test	Expected	Got	
✓	`print(STRtoINT('zero'))`	`0`	`0`	✓
✓	`print(STRtoINT('one'))`	`1`	`1`	✓

Passed all tests! ✓

Correct

Marks for this submission: 6.67/6.67.

Question 5

Correct

Mark 6.67 out of 6.67

Read a string from the user and **return** the integer conversion of it. Ensure the conversion is successful by removing any non-numeric characters. You may assume that the input will contain at least 1 numeric character.

For example:

Test	Input	Result
`print(whatNumber())`	`he3llo0`	`30`

Answer: (penalty regime: 0 %)

Reset answer

```
def whatNumber():
    strng= input()
    aval=""
    for x in strng:
        if x.isdigit():
            aval= aval + x


    y = int(aval)

    return y
```

	Test	Input	Expected	Got	
✓	`print(whatNumber())`	`he3llo0`	`30`	`30`	✓
✓	`print(whatNumber())`	`b7y6e3`	`763`	`763`	✓

Passed all tests! ✓

Correct

Marks for this submission: 6.67/6.67.

Question 6

Correct

Mark 6.67 out of 6.67

Given a full first name, middle name, last name, and domain, **print** to the screen their email address. The address should take the form:

> <first>.<middle initial>.<last>@<domain>.com
> Example: john.m.smilth@coolguy.com

- Only include a middle initial (the first letter of the middle name).
- Ensure the address is all lowercase.
- Append `'.com'` to the given domain.

For example:

Test	Result
`q6('jane','donna','smith','google')`	`jane.d.smith@google.com`

Answer: (penalty regime: 0 %)

Reset answer

```
def q6(first,middle,last,domain):
    x = f"{first.lower()}.{middle[0].lower()}.{last.lower()}@{domain.lower()}.com"
    print(x)
    pass

```

	Test	Expected	Got	
✓	`q6('jane','donna','smith','google')`	`jane.d.smith@google.com`	`jane.d.smith@google.com`	✓
✓	`q6('arron','luke','brown','google')`	`arron.l.brown@google.com`	`arron.l.brown@google.com`	✓

Passed all tests! ✓

Correct

Marks for this submission: 6.67/6.67.

Question 7

Correct

Mark 6.67 out of 6.67

Copy the contents of the file whose filename is given in `infile` to the file whose name is given in `outfile`. Overwrite `outfile` if it already exists.

infile will use users.txt as a source file of contents to copy into outfile

For example:

Test	Result

`q7('users.txt','this.txt')`
`def check(outfile):`
`  with open(outfile) as of:`
`  content = of.read()`
`  print(content)`
`check('this.txt')`
	`john.doe`
	`jane.smith`
	`adam.jones`
	`emily.brown`
	`michael.wilson`
	`alice.smilth`
	`bob.johnson`
	`sara.williams`
	`david.miller`
	`lisa.taylor`
	`kevin.jackson`
	`hannah.anderson`
	`peter.davis`
	`olivia.martin`
	`jack.wilson`
	`grace.thompson`
	`samuel.roberts`
	`natalie.white`
	`jacob.hall`
	`mia.carter`

Answer: (penalty regime: 0 %)

Reset answer

```
def q7(infile,outfile):
    pass
    with open(infile, 'r') as f_read:
        contents = f_read.read()

    with open(outfile, 'w') as f_write:
        f_write.write(contents)

```

	Test	Expected	Got	

	Test	Expected	Got	

✓	q7('users.txt','this.txt')
	def check(outfile):
	    with open(outfile) as of:
	        content = of.read()
	    print(content)
	check('this.txt')
		john.doe
		jane.smith
		adam.jones
		emily.brown
		michael.wilson
		alice.smilth
		bob.johnson
		sara.williams
		david.miller
		lisa.taylor
		kevin.jackson
		hannah.anderson
		peter.davis
		olivia.martin
		jack.wilson
		grace.thompson
		samuel.roberts
		natalie.white
		jacob.hall
		mia.carter
			john.doe
			jane.smith
			adam.jones
			emily.brown
			michael.wilson
			alice.smilth
			bob.johnson
			sara.williams
			david.miller
			lisa.taylor
			kevin.jackson
			hannah.anderson
			peter.davis
			olivia.martin
			jack.wilson
			grace.thompson
			samuel.roberts
			natalie.white
			jacob.hall
			mia.carter
				✓

✓	q7('file1.txt','this.txt')
	def check(outfile):
	    with open(outfile) as of:
	        content = of.read()
	    print(content)
	check('this.txt')
		Why do programmers prefer dark mode?
		Because light attracts bugs!
		I would tell you a joke about UDP, but you might not get it.
		Why did the programmer quit his job?
		Because he didn't get arrays.
		I told my wife she should embrace her mistakes. She gave me a hug.
		Debugging: Removing the needles from the haystack.
		Why did the developer go broke? Because he used up all his cache.
		Why don't programmers like nature? It has too many bugs.
		There are only 10 types of people in the world: those who understand binary, and those who don't.
		Why was the JavaScript developer sad?
		Because he didn't know how to 'null' his emotions.
		Why do Java developers wear glasses?
		Because they can't C#.
			Why do programmers prefer dark mode?
			Because light attracts bugs!
			I would tell you a joke about UDP, but you might not get it.
			Why did the programmer quit his job?
			Because he didn't get arrays.
			I told my wife she should embrace her mistakes. She gave me a hug.
			Debugging: Removing the needles from the haystack.
			Why did the developer go broke? Because he used up all his cache.
			Why don't programmers like nature? It has too many bugs.
			There are only 10 types of people in the world: those who understand binary, and those who don't.
			Why was the JavaScript developer sad?
			Because he didn't know how to 'null' his emotions.
			Why do Java developers wear glasses?
			Because they can't C#.
				✓

Passed all tests! ✓

Correct

Marks for this submission: 6.67/6.67.

Question 8

Correct

Mark 6.67 out of 6.67

Given an email address of the form:

```
<first>.<middle initial>.<last>@<domain>.com
```

return the 4 elements of the address as a tuple in the order that they appear in the address.

For example, if given `'nicholas.r.yost@somedomain.com`,

`('nicholas','r','yost','somedomain')` should be returned.

For example:

Test	Result
`print(q8('jane.d.smith@google.com'))`	`('jane', 'd', 'smith', 'google')`

Answer: (penalty regime: 0 %)

Reset answer

```
def q8(address):
    x = address.replace('@', '.').split('.')
    x.pop()
    z = tuple(x)
    return z

```

	Test	Expected	Got	
✓	`print(q8('jane.d.smith@google.com'))`	`('jane', 'd', 'smith', 'google')`	`('jane', 'd', 'smith', 'google')`	✓
✓	`print(q8('sarah.h.hank@yahoo.com'))`	`('sarah', 'h', 'hank', 'yahoo')`	`('sarah', 'h', 'hank', 'yahoo')`	✓

Passed all tests! ✓

Correct

Marks for this submission: 6.67/6.67.

Question 9

Correct

Mark 6.67 out of 6.67

Given a string, **return** a dictionary whose keys are the set of unique characters within the string and whose values are the count of occurrences of each character.
For example, if given `'hello'`, the returned dictionary should be `{ 'l':2, 'h':1, 'e':1, 'o':1 }`
collections.Counter has been disabled for this function.

For example:

Test	Result
`print(q1('jello'))`	`{'j': 1, 'e': 1, 'l': 2, 'o': 1}`

Answer: (penalty regime: 0 %)

Reset answer

```
def q1(strng):
    dictionary = dict()
    for letter in strng:
        dictionary[letter] = strng.count(letter)
    return dictionary

```

	Test	Expected	Got	
✓	`print(q1('jello'))`	`{'j': 1, 'e': 1, 'l': 2, 'o': 1}`	`{'j': 1, 'e': 1, 'l': 2, 'o': 1}`	✓

Passed all tests! ✓

Correct

Marks for this submission: 6.67/6.67.

Question 10

Correct

Mark 6.67 out of 6.67

Return a tuple consisting of the following elements in the order listed:

1. your last name as a `string`
2. your last name reversed as a `string`

Ensure the second element retains the same case as the first

For example:

Test	Result
`print(q10('john'))`	`('john', 'nhoj')`

Answer: (penalty regime: 0 %)

Reset answer

```
def q10(name):
    strng = str(name)
    x = strng[::-1]
    y = (strng, x)
    return tuple(y)

    pass

```

	Test	Expected	Got	
✓	`print(q10('john'))`	`('john', 'nhoj')`	`('john', 'nhoj')`	✓
✓	`print(q10('Smith'))`	`('Smith', 'htimS')`	`('Smith', 'htimS')`	✓

Passed all tests! ✓

Correct

Marks for this submission: 6.67/6.67.

Question 11

Correct

Mark 6.67 out of 6.67

Write the function `dict_ports` that accepts a dictionary where the keys are protocol names and the values are port numbers. Iterate through the keys and values and **print** a formatted string at the end of each iteration. The formatted string should follow the format of `"Port: ___ is associated with ___"` where the blanks are filled with the appropriate key or value.

Example:

```
dict_ports({'SSH' : 22})
```

Prints:

```
Port: 22 is associated with SSH
```

Example2:

```
dict_ports({'SSH':22, 'SMTP':25})
```

Prints:

```
Port: 22 is associated with SSH
Port: 25 is associated with SMTP
```

For example:

Test	Result

`dict_ports({'SSH':22, 'SMTP':25, "DNS":53, 'HTTP':80})`	`Port: 22 is associated with SSH`
	`Port: 25 is associated with SMTP`
	`Port: 53 is associated with DNS`
	`Port: 80 is associated with HTTP`

Answer: (penalty regime: 0 %)

Reset answer

```
def dict_ports(ports):
    for key, value in ports.items():
        print(f'Port: {value} is associated with {key}')

    pass
```

	Test	Expected	Got	

	Test	Expected	Got	

✓	`dict_ports({'SSH':22, 'SMTP':25, "DNS":53, 'HTTP':80})`	`Port: 22 is associated with SSH`
		`Port: 25 is associated with SMTP`
		`Port: 53 is associated with DNS`
		`Port: 80 is associated with HTTP`
			`Port: 22 is associated with SSH`
			`Port: 25 is associated with SMTP`
			`Port: 53 is associated with DNS`
			`Port: 80 is associated with HTTP`
				✓

✓	`dict_ports({"DNS":53, "SSH":22})`	`Port: 53 is associated with DNS` `Port: 22 is associated with SSH`	`Port: 53 is associated with DNS` `Port: 22 is associated with SSH`	✓

✓	`dict_ports({'Telnet':23, 'LDAP':389, "POP3":110, 'TFTP':69})`	`Port: 23 is associated with Telnet`
		`Port: 389 is associated with LDAP`
		`Port: 110 is associated with POP3`
		`Port: 69 is associated with TFTP`
			`Port: 23 is associated with Telnet`
			`Port: 389 is associated with LDAP`
			`Port: 110 is associated with POP3`
			`Port: 69 is associated with TFTP`
				✓

Passed all tests! ✓

Correct

Marks for this submission: 6.67/6.67.

Question 12

Correct

Mark 6.67 out of 6.67

Return the sum of all arguments (both positional and keyword). You may assume the values of all arguments are integers.

Example:

```
q1(10, 20, 30, a = 5, b = 9, c = 20)
```

Should return: `94`

For example:

Test	Result
`print(q1(50, 70, 30, a = 5, b = 9, c = 20))`	184

Answer: (penalty regime: 0 %)

Reset answer

```
def q1(*args,**kwargs):
    return sum(args) + sum(kwargs.values())
    pass
```

	Test	Expected	Got	
✓	`print(q1(50, 70, 30, a = 5, b = 9, c = 20))`	184	184	✓

Passed all tests! ✓

Correct

Marks for this submission: 6.67/6.67.

Question 13

Correct

Mark 6.67 out of 6.67

Use the appropriate options to build a function that creates a client socket to connect to a server. The client socket you are building should send a message in bytes to the server as well as receive a message in bytes from the server.

```
import [socket] ✓
[def] ✓ client_socket(address, port):
    s = socket.[socket()] ✓
  s.connect[((address, port))] ✓
  mesg = [b] ✓ "It's me, Mario!"
  s.[sendall] ✓ (mesg)
    chunk = s.[recv] ✓ (10)
    while chunk:
        msg.[extend] ✓ (chunk)
        chunk = s.recv(10)
    print(msg)
```

[network] [server]

[class] [make]

[connect()] [bind()]

[(port, address)] [(address, port)] [(address)(port)]

[f] [B] [F]

[recv] [transmit] [sendto]

[send] [receive] [get]

[append] [add] [save]

Your answer is correct.

Question 14

Correct

Mark 6.67 out of 6.67

Create a class with specific functions

- Create a class named `Car`.
- `Car` instances should store the current `speed` of the `Car.`
- An additional function `getspeed` should return the current speed.
- Function `speedup` should increase speed by 1
- Function `slowdown` should decrease speed by 1
- Function `stop` should set the speed to 0
- The class should also respond to a conversion to string by returning the string `'Current speed: '` followed by the current speed.

For example:

Test	Result

`test_car = Car()`
`print(test_car)`
`test_car.speedup()`
`print(test_car)`
`test_car.slowdown()`
`print(test_car)`
`test_car.stop()`
`print(test_car)`
`print(test_car.getspeed())`
	`Current speed: 0`
	`Current speed: 1`
	`Current speed: 0`
	`Current speed: 0`
	`0`

Answer: (penalty regime: 0 %)

Reset answer

```
#No starting code provided

class Car:
    def __init__(self):
        self.speed = 0
    def getspeed(self):
        return self.speed
    def speedup(self):
        self.speed = self.speed + 1
    def slowdown(self):
        self.speed = self.speed - 1
    def stop (self):
        self.speed = 0
    def __str__(self):
        return f'Current speed: {self.speed}'
```

	Test	Expected	Got	

	Test	Expected	Got	

✓	`test_car = Car()`
	`print(test_car)`
	`test_car.speedup()`
	`print(test_car)`
	`test_car.slowdown()`
	`print(test_car)`
	`test_car.stop()`
	`print(test_car)`
	`print(test_car.getspeed())`
		`Current speed: 0`
		`Current speed: 1`
		`Current speed: 0`
		`Current speed: 0`
		`0`
			`Current speed: 0`
			`Current speed: 1`
			`Current speed: 0`
			`Current speed: 0`
			`0`
				✓

✓	`test_car = Car()`
	`print(test_car)`
	`test_car.speedup()`
	`print(test_car)`
	`test_car.speedup()`
	`print(test_car)`
	`test_car.slowdown()`
	`print(test_car)`
	`test_car.stop()`
	`print(test_car)`
	`print(test_car.getspeed())`
		`Current speed: 0`
		`Current speed: 1`
		`Current speed: 2`
		`Current speed: 1`
		`Current speed: 0`
		`0`
			`Current speed: 0`
			`Current speed: 1`
			`Current speed: 2`
			`Current speed: 1`
			`Current speed: 0`
			`0`
				✓

Passed all tests! ✓

After reviewing the code, the set speed is adding the speed from the parameter on top of the existing speed, The naming convention when creating a class in Python follows the CamelCase style, Car should be capitalized.

Correct

Marks for this submission: 6.67/6.67.

Question 15

Correct

Mark 6.67 out of 6.67

Write a Python function that receives an IP address as a string parameter that checks to see if the address is a class A, B or C and returns the Class IP range that it belongs to. The function should **return** a string that is formatted in the same way as the example shown below. (Take note of capitalization and punctuation!)

Example:

```
check_ip_address("10.10.10.1")
```

should return the following string:

```
10.10.10.1 belongs to Class A.
```

Note: Class A - 10.0.0.0 - 10.255.255.255
Class B - 172.16.0.0 - 172.31.255.255
Class C - 192.168.0.0 - 192.168.255.255

Extra note: For the purposes of this question you only need to check the first octet.

For example:

Test	Result
`print(check_ip_address("10.10.10.1"))`	`10.10.10.1 belongs to Class A.`

Answer: (penalty regime: 0 %)

Reset answer

```
def check_ip_address(ip_address):
    x = int(ip_address.split('.')[0])
    if x in range(10,11):
        return f'{ip_address} belongs to Class A.'
    elif x in range(172,173):
        return f'{ip_address} belongs to Class B.'
    elif x in range(192,193):
        return f'{ip_address} belongs to Class C.'
```

	Test	Expected	Got	
✓	`print(check_ip_address("10.10.10.1"))`	`10.10.10.1 belongs to Class A.`	`10.10.10.1 belongs to Class A.`	✓
✓	`print(check_ip_address("172.16.0.1"))`	`172.16.0.1 belongs to Class B.`	`172.16.0.1 belongs to Class B.`	✓
✓	`print(check_ip_address("192.168.1.1"))`	`192.168.1.1 belongs to Class C.`	`192.168.1.1 belongs to Class C.`	✓

Passed all tests! ✓

Correct

Marks for this submission: 6.67/6.67.