**ICS4U Course Feedback**

|  |
| --- |
| **Data Structures and Algorithms** |
| *Students design, create, test, and analyze complex algorithms and data structure through the creation of computer programs with clear documentation to enhance their logical thinking skills.* |

*IDQ = insufficient data or quality (less than 50%), B = beginning (50-59%), D = developing (60-69%), P = proficient (70-79%), C = comprehensive (80-89%), E = exemplary (90-100%)*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Data Structures and Algorithms*  *RVN = Review and New, DST = Data Structures, EDS = Extended Data Structures, ALG = Algorithms, REC = Recursion (1 = feedback submission, 2 = final submission)* | | | | | | |
| *We are learning to work with data types and proper code maintenance techniques (A1, A4)* | | | | | | |
| Success Criteria (I can…) | IDQ | B | D | P | C | E |
| Create & use integer division and resultant remainders in computer programs. | * RVN | * RVN | * RVN | * RVN | * RVN | * RVN |
| complete type conversions. | * RVN | * RVN | * RVN | * RVN | * RVN | * RVN |
| work independently, using support documentation, to resolve syntax issues during software development. | * RVN * DST | * RVN * DST | * RVN * DST | * RVN * DST | * RVN * DST | * RVN * DST |
| create fully documented program code according to industry standards. | * DST * EDS | * DST * EDS | * DST * EDS | * DST * EDS | * DST * EDS | * DST * EDS |
| use one-dimensional arrays of compound data types (i.e. objects) | * EDS * ALG | * EDS * ALG | * EDS * ALG | * EDS * ALG | * EDS * ALG | * EDS * ALG |
| *We are learning to design and apply modular programming concepts including complex data types (A2, C1)* | | | | | | |
| Success Criteria (I can…) | IDQ | B | D | P | C | E |
| create a modular program that is divided among multiple files. | * DST | * DST | * DST | * DST | * DST | * DST |
| modify existing modular program code to enhance the functionality of the program. | * EDS | * EDS | * EDS | * EDS | * EDS | * EDS |
| decompose a problem into modules, classes, or abstract data types using an object-oriented design methodology (using classes, methods, etc. using UML, etc.). | * DST * EDS | * DST * EDS | * DST * EDS | * DST * EDS | * DST * EDS | * DST * EDS |
| apply the principle of reusability in program design (use of inheritance). | * EDS | * EDS | * EDS | * EDS | * EDS | * EDS |
| *We are learning to design, write and analyze complex algorithms and subprograms (A3, C2)* | | | | | | |
| Success Criteria (I can…) | IDQ | B | D | P | C | E |
| read from, and write to, an external file (i.e. database, API, text file, binary file, etc.). | * DST * EDS | * DST * EDS | * DST * EDS | * DST * EDS | * DST * EDS | * DST * EDS |
| compare the efficiency of **sorting** algorithms, using run times and computational complexity analysis. | * ALG | * ALG | * ALG | * ALG | * ALG | * ALG |
| compare the efficiency of linear and binary **searches**, using run times and computational complexity analysis. | * ALG | * ALG | * ALG | * ALG | * ALG | * ALG |
| identify common pitfalls while developing recursive functions. | * REC1 * REC2 | * REC1 * REC2 | * REC1 * REC2 | * REC1 * REC2 | * REC1 * REC2 | * REC1 * REC2 |
| design a simple and efficient recursive algorithm. | * REC1 * REC2 | * REC1 * REC2 | * REC1 * REC2 | * REC1 * REC2 | * REC1 * REC2 | * REC1 * REC2 |
| visualize how the recursive algorithm works narratively. | * REC1 * REC2 | * REC1 * REC2 | * REC1 * REC2 | * REC1 * REC2 | * REC1 * REC2 | * REC1 * REC2 |
|  | | | | | | |
| **Overall teacher comments:** | | | | | | |
| **Student unit reflection or comments:** | | | | | | |