Introduction to Python

Course Number: 2000

Course Level: Standard/College Bound

Grade: 9-12

Time/Credit: 83 minutes daily: Grades 9, 10, 11, and 12

.5 credits

Development Date: Fall 2025 - Spring 2026

Board Approval Date: _____

Central Bucks School District
20 Welden Drive
Doylestown, PA 18901

Table of Contents

Central Bucks School District
K-12 Computer Science Teaching and Learning Principles

1. Investigating, problem-solving, and communicating about computer science (speaking, reading, writing, representing, and listening) stimulates students' curiosity, builds confidence, and leads students toward deeper conceptual understanding.

2. Lessons should address students' strengths, interests, and learning styles through differentiated instruction.

3. All students can achieve excellence in computer science when:
    - instruction is appropriately paced
    - teachers set high expectations
    - students feel supported by their teachers, parents and administrators

4. Students must be actively engaged in learning experiences designed to connect to their prior knowledge.

5. For students to transfer and use knowledge meaningfully teachers need to:
    - present new ideas within rich and meaningful contexts
    - help students to make connections to their lives both within and outside of school

6. Computer Science instruction is more effective when teachers incorporate:
    - manipulatives
    - integrated development environments, both online and device based

7. Assessment should be ongoing, diagnostic, and aligned with instruction.

Documents Considered:

Computer Science Teachers Association (2017). CSTA K-12 Computer Science Standards, Revised 2017. Retrieved from http://www.csteachers.org/standards.

Kilpatrick, J., Swafford, J., & Findell, B. (2001). *Adding It Up: Helping Children Learn Mathematics.* Washington, DC: National Academy Press.

Marzano, R., & Pickering, D. (1997). *Dimensions of Learning.* Alexandria,VA: Association for Supervision and Curriculum Development; Aurora, CO: Mid-continent Regional Educational Laboratory.

Tomlinson, Carol Ann. (1999). *The Differentiated Classroom: Responding to the Needs of All Learners.* Alexandria, VA: Association for Supervision and Curriculum Development.

Zemelman, S., Daniels, H., & Hyde, A. (1998). *Best practice: New Standards for Teaching and Learning in America's Schools*. Portsmouth, NH: Heinemann.

Central Bucks School District
K-12 Computer Science Enduring Understandings

Students will understand that:

Computing Systems

- Abstractions hide the underlying implementation details of computing systems embedded in everyday objects.
- Levels of abstraction and interactions between application software, system software, and hardware layers can be compared.
- Guidelines that convey systematic troubleshooting strategies can be developed that others can use to identify and fix errors.

Networks and the Internet

- There is scalability and reliability of networks, in which the relationship between routers, switches, servers, topology, and addressing can be described.
- Sensitive data can be affected by malware and other attacks.
- Security measures can address various scenarios based on factors such as efficiency, feasibility, and ethical impacts.

Data and Analysis

- There are different bit representations of real-world phenomena, such as characters, numbers, and images.
- There are tradeoffs in how data elements are organized and where data is stored.

Algorithms and Programming

- Algorithms are used to solve computational problems by leveraging prior student knowledge and personal interests.
- Programmers must justify the selection of specific control structures when tradeoffs involve implementation, readability, and program performance, and explain the benefits and drawbacks of choices made.
- Problems can be decomposed into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.
- Documentation of design decisions using text, graphics, presentations, and/or demonstrations is an integral part in the development of complex programs.

Impacts of Computing

- Computing affects personal, ethical, social, economic, and cultural practices.
- Computational artifacts must be tested and refined to reduce bias and equity deficits.

Central Bucks School District
K-12 Computer Science Program Essential Questions

Computing Systems

- What is an abstraction?
- How does abstraction manage complexity?

Networks and the Internet

- How do scalability and redundancy relate to reliability in the Internet?
- What is malware and how is sensitive data affected by it?
- How do efficiency, feasibility, and ethical impacts impact security measures?

Data and Analysis

- What are the tradeoffs in how data elements are organized and where data is stored?
- How do we collect, visualize, and store large amounts of data?
- Why do we create computational models and how does the data we use affect those models?
- What is the purpose of creating prototypes to solve computational problems?

Algorithms and Programming

- What is an algorithm?
- How do algorithms and programs differ?
- Why is the selection of specific control structures important when considering implementation, readability, and program performance?
- Why is the decomposition into smaller components in constructs such as procedures, modules, and/or objects important in the software development process?
- Why is documentation and design decisions an integral part in the development of complex programs?

Impacts of Computing

- What is the purpose of collecting data?
- How is data used or abused?
- What are the ways that computing affects personal, ethical, social, economic, and cultural practices?
- Why must we test and refine computational artifacts to remove to reduce bias and equity deficits?

Central Bucks School District

Introduction to Python Overview

Desired Results

Course Description: 2000 Introduction to Python (9 weeks, .5 credits)

Introduction to Python introduces students to the fundamentals of computer programming, with an emphasis on helping students develop logical thinking and problem-solving skills. Students begin by learning to design, code, and test their Python programs while applying mathematical concepts. Students then move to more advanced programming concepts and learn to create more powerful programs using functions, strings, data structures, file I/O operations, and objects, as time permits. Students planning to take the Advanced Placement Computer Science A Course in the future should select this course. Up to one credit from Computer Science courses may be used to fill your graduation requirements for mathematics.

Prerequisite: None.

Committee Members: Amy MacMinn, Aaron Fromal, Jenn Robinson, Matthew Quinn, Chris McGlone, Anthony Paglione, Brian Novick – Mathematics Supervisor

Course Grading: The final course grade is calculated as follows:

- Each marking period grade counts for 50% of the students' course grade.
- This course is an elective. There is no final exam.

Introduction to Python Pacing Guide

Content	Blocks
Unit 1: Intro to Programming with Graphics	**8**
Introduction to Computer Science Paradigms: • Sequencing • Selection • Iteration • Program Documentation	7
Unit Assessment	1
Unit 2: Introduction to Python Basics	**9**
• Printing • Variables • User Input • Converting Input Types • Arithmetic Expressions • String Operators	8
Unit Assessment	1
Unit 3: Conditionals	**4**
• If Statements • Boolean Values • Logical Operators • Comparison Operators • Floating Point Numbers and “Equality”	3
Unit Assessment(s)	1
Unit 4: Loops	**4**
• While Loops • For Loops • Break and Continue • Nested Control Structures	3
Unit Assessment	1
Unit 5: Functions	**5**
• Functions • Namespaces • Parameters • Return Values • Exceptions	4
Unit Assessment	1
Unit 6: Strings	**5**
• Indexing and Slicing • Math Operators on Strings • For Loops Over a String • String Methods	4
Unit Assessment	1
Unit 7: Data Structures	**5**
• Tuples • Lists • For Loops and Lists	4

• List Methods	
Unit Assessment	1
Unit 8: Artificial Intelligence (Time Permitting)	**5**
• Machine Learning • Neural Networks • Prompt Architecture	5
Total Days for Course:	**45**

Required and Supplementary Texts/Software

- Student Laptops
- Recommended Integrated Development Environments
    - CodeHS – Introduction to Computer Science with Python 3
    - VisualStudio for Education

Central Bucks School District – Course of Study

Unit 1: Introduction to Programming with Graphics

Desired Results

Essential Questions:

- How can visual programming with graphics build strong habits for problem solving and algorithm design?
- How do sequencing, loops, conditionals, and functions work together to produce meaningful visual output?
- Why are decomposition, naming, and comments important when creating programs other people can read and revise?
- How does user input make a program more interactive and flexible?

Enduring Understandings:

Students will understand that…

- Programs are precise sequences of instructions that can be combined into larger algorithms to solve problems.
- Complex problems become more manageable when they are decomposed into smaller functions or subtasks.
- Loops and conditionals allow programs to repeat actions and respond to changing conditions efficiently.
- Clear naming, comments, and visual testing support readable and maintainable code.

Knowledge:

Students will know…

- Basic graphics commands and coordinate-based movement
- The role of sequencing, selection, and iteration in visual programs
- Functions, parameters, top-down design, and decomposition
- Variables, user input, comments, and naming conventions
- Combining multiple control structures to solve larger drawing problems

Skills:

Students will be able to…

- Write Python programs that use Turtle Graphics commands to produce intended drawings.
- Use coordinates, angles, loops, and conditionals to control visual output.
- Break a larger graphics task into smaller functions with meaningful names.
- Incorporate comments and user input to improve readability and interactivity.
- Test and revise graphical programs for correctness and visual accuracy.

Standards:

3A-AP-13	Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests.
3A-AP-15	Justify the selection of specific control structures when tradeoffs involve implementation, readability, and program performance, and explain the benefits and drawbacks of choices made.
3A-AP-17	Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.
3A-AP-23	Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs.

Acceptable Evidence

Acceptable evidence of learning other than Performance Tasks:

Formative Assessment

- Warm-ups
- Check point assignments
- Read and Take Notes (RTN)
- Class work
- Homework
- Exit slips
- Keep a notebook
- Quizzes
- Programming Assignments
- Unit Quiz

Summative Assessment

- Unit Assessment

Suggested learning experiences and instruction

Learning Experiences:

- Use of projector to demonstrate new material.
- Use of Power Point slides to present concepts and terminology.

Instructional Strategies:

- Emphasize the steps in problem solving and the use of algorithms.
- Emphasize the use of algorithms in problem solving.
- Emphasize the difference between a compiler and an interpreter.
- Have students compile and run a basic Python program.
- Complete teacher-generated self-review questions, multiple-choice, true/false, and short answer problems.

Differentiation:

- Work with students in small groups to intervene or enrich as needed
- Provide additional scaffolds for emerging learners.
- Allow on level learners to modify existing code and explain their changes.
- Provide opportunities for advanced learners to add features, variables, or additional functionality.

For Language Development:

- Create a student-made dictionary that includes terms, definitions, and examples organized by chapter.
- Make flash cards for terms and examples, then mix and match.
- Create study guides with clear definitions and vocabulary development opportunities.

For Advanced Learners:

- Meet in small groups with students who need enrichment to provide instruction and learning activities on the optional skills followed up by an alternative and more complex assignment.

- Excuse the students from standard practice and use appropriately challenging practice that requires students to apply the concepts in more rigorous ways to challenge their thinking and allow students to go deeper on the same skills.
- Provide more rigorous opportunities for application of skills including enrichment projects.

Central Bucks School District – Course of Study

Unit 2: Introduction to Python Basics

Desired Results

Essential Questions:

- How do variables, values, and data types allow a Python program to represent information?
- How do input, processing, and output work together in a useful console program?
- Why do programmers need to understand both numeric and string operations?
- How does documentation improve the usability and readability of programs?

Enduring Understandings:

Students will understand that…

- Variables store information so a program can reuse, transform, and display data.
- Programs process user input through expressions and produce output for a purpose.
- Different data types support different operations and lead to different program behavior.
- Comments help programmers communicate intent and support debugging and revision.

Knowledge:

Students will know…

- Printing and console output
- Variables, assignment, identifiers, and data types
- User input and type conversion
- Mathematical operators, precedence, and simple computations
- String operators and string concatenation
- Comments and readable program structure

Skills:

Students will be able to…

- Write console programs that gather input and display formatted output.
- Declare, initialize, and update variables appropriately.
- Use arithmetic and string operations to solve introductory programming problems.
- Convert user input into the type needed for a computation.
- Document simple programs with appropriate comments.

Standards:

3A-AP-13	Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests.
3A-AP-17	Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.
3A-AP-23	Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs.

Acceptable Evidence

Acceptable evidence of learning other than Performance Tasks:

Formative Assessment

- Warm-ups
- Check point assignments

- Read and Take Notes (RTN)
- Class work
- Homework
- Exit slips
- Keep a notebook
- Quizzes
- Programming Assignments
- Unit Quiz

Summative Assessment

- Unit Assessment

Suggested learning experiences and instruction

Learning Experiences:

- Use of projector to demonstrate new material.
- Use of Power Point slides to present concepts and terminology.

Instructional Strategies:

- Emphasize the use of algorithms in problem solving.
- Have students enter, compile, and run simple Python programs.
- Complete teacher-generated self-review questions, multiple-choice, true/false, and short answer problems.

Differentiation:

- Work with students in small groups to intervene or enrich as needed
- Provide additional scaffolds for emerging learners.
- Allow on level learners to modify existing code and explain their changes.
- Provide opportunities for advanced learners to add features, variables, or additional functionality.

For Language Development:

- Create a student-made dictionary that includes terms, definitions, and examples organized by chapter.
- Make flash cards for terms and examples, then mix and match.
- Create study guides with clear definitions and vocabulary development opportunities.

For Advanced Learners:

- Meet in small groups with students who need enrichment to provide instruction and learning activities on the optional skills followed up by an alternative and more complex assignment.
- Excuse the students from standard practice and use appropriately challenging practice that requires students to apply the concepts in more rigorous ways to challenge their thinking and allow students to go deeper on the same skills.
- Provide more rigorous opportunities for application of skills including enrichment projects.

Central Bucks School District – Course of Study

Unit 3: Conditionals

Desired Results

Essential Questions:

- How do programs make decisions based on the data they receive?
- What is the difference between a boolean condition and the action that follows from it?
- How do comparison and logical operators help us express more complex decisions?
- Why must programmers be cautious when comparing floating-point values?

Enduring Understandings:

Students will understand that…

- Conditionals allow a program to follow different paths based on evaluated conditions.
- Boolean expressions represent true/false logic that drives program decisions.
- Logical combinations of conditions can model more realistic decision-making.
- Precision and representation affect how computers compare some numeric values.

Knowledge:

Students will know…

- Booleans and truth values
- If statements and if/else structures
- Comparison operators and logical operators
- Floating-point values, rounding, and equality concerns

Skills:

Students will be able to…

- Write conditional statements that correctly branch program flow.
- Evaluate boolean expressions and predict program behavior.
- Use comparison and logical operators to combine multiple conditions.
- Recognize and address issues that arise when comparing floating-point values.
- Revise programs to improve clarity and correctness of decision logic.

Standards:

3A-AP-13	Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests.
3A-AP-15	Justify the selection of specific control structures when tradeoffs involve implementation, readability, and program performance, and explain the benefits and drawbacks of choices made.
3A-AP-21	Evaluate and refine computational artifacts to make them more usable and accessible.
3A-AP-23	Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs.

Acceptable Evidence

Acceptable evidence of learning other than Performance Tasks:

Formative Assessment

- Warm-ups

- Check point assignments
- Read and Take Notes (RTN)
- Class work
- Homework
- Exit slips
- Keep a notebook
- Quizzes
- Programming Assignments
- Unit Quiz

Summative Assessment

- Unit Assessment

Suggested learning experiences and instruction

Learning Experiences:

- Use of projector to demonstrate new material.
- Use of Power Point slides to present concepts and terminology.

Instructional Strategies:

- Emphasize the use of algorithms in problem solving.
- Complete teacher-generated self-review questions, multiple-choice, true/false, and short answer problems.

Differentiation:

- Work with students in small groups to intervene or enrich as needed
- Provide additional scaffolds for emerging learners.
- Allow on level learners to modify existing code and explain their changes.
- Provide opportunities for advanced learners to add features, variables, or additional functionality.

For Language Development:

- Create a student-made dictionary that includes terms, definitions, and examples organized by chapter.
- Make flash cards for terms and examples, then mix and match.
- Create study guides with clear definitions and vocabulary development opportunities.

For Advanced Learners:

- Meet in small groups with students who need enrichment to provide instruction and learning activities on the optional skills followed up by an alternative and more complex assignment.
- Excuse the students from standard practice and use appropriately challenging practice that requires students to apply the concepts in more rigorous ways to challenge their thinking and allow students to go deeper on the same skills.
- Provide more rigorous opportunities for application of skills including enrichment projects.

Central Bucks School District – Course of Study

Unit 4: Loops

Desired Results

Essential Questions:

- How do loops help programmers write more efficient and generalizable code?
- When is a for loop more appropriate than a while loop?
- How do break and continue affect the flow of a loop?
- Why do nested control structures increase both power and complexity in a program?

Enduring Understandings:

Students will understand that…

- Loops support efficiency by automating repeated behavior in a program.
- Different loop structures are appropriate for different programming situations.
- Loop control statements influence readability, termination, and program behavior.
- Nested control structures require careful design, testing, and tracing.

Knowledge:

Students will know…

- While loops and for loops
- Loop conditions and iteration variables
- Break and continue
- Nested loops and nested control structures
- Common loop errors such as infinite loops and incorrect boundaries

Skills:

Students will be able to…

- Write while and for loops that repeat behavior correctly.
- Trace loops to determine how many times code executes and why.
- Use break and continue strategically when appropriate.
- Combine loops with conditionals to solve more complex problems.
- Debug loop logic and refine code for readability and correctness.

Standards:

3A-AP-15	Justify the selection of specific control structures when tradeoffs involve implementation, readability, and program performance, and explain the benefits and drawbacks of choices made.
3A-AP-17	Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.
3A-AP-21	Evaluate and refine computational artifacts to make them more usable and accessible.
3A-AP-23	Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs.

Acceptable Evidence

Acceptable evidence of learning other than Performance Tasks:

Formative Assessment

- Warm-ups

- Check point assignments
- Read and Take Notes (RTN)
- Class work
- Homework
- Exit slips
- Keep a notebook
- Quizzes
- Programming Assignments
- Unit Quiz

Summative Assessment

- Unit Assessment

Suggested learning experiences and instruction

Learning Experiences:

- Use of projector to demonstrate new material.
- Use of Power Point slides to present concepts and terminology.

Instructional Strategies:

- Emphasize the use of algorithms in problem solving.
- Complete teacher-generated self-review questions, multiple-choice, true/false, and short answer problems.

Differentiation:

- Work with students in small groups to intervene or enrich as needed
- Provide additional scaffolds for emerging learners.
- Allow on level learners to modify existing code and explain their changes.
- Provide opportunities for advanced learners to add features, variables, or additional functionality.

For Language Development:

- Create a student-made dictionary that includes terms, definitions, and examples organized by chapter.
- Make flash cards for terms and examples, then mix and match.
- Create study guides with clear definitions and vocabulary development opportunities.

For Advanced Learners:

- Meet in small groups with students who need enrichment to provide instruction and learning activities on the optional skills followed up by an alternative and more complex assignment.
- Excuse the students from standard practice and use appropriately challenging practice that requires students to apply the concepts in more rigorous ways to challenge their thinking and allow students to go deeper on the same skills.
- Provide more rigorous opportunities for application of skills including enrichment projects.

Central Bucks School District – Course of Study

Unit 5: Functions

Desired Results

Essential Questions:

- How do functions help programmers organize, reuse, and test code?
- What is the relationship among parameters, local scope, and return values?
- Why is decomposition essential when programs become larger or more complex?
- How do exceptions reveal assumptions and possible failure points in a program?

Enduring Understandings:

Students will understand that…

- Functions allow programmers to group related steps into reusable units.
- Parameters and return values define the input-output behavior of a function.
- Scope affects which variables are available in different parts of a program.
- Exceptions and errors are opportunities to test assumptions and improve reliability.

Knowledge:

Students will know…

- Function definitions and calls
- Parameters, arguments, and return values
- Local scope and namespaces
- Exceptions and common runtime errors

Skills:

Students will be able to…

- Define and call functions with and without parameters.
- Write functions that return useful values to other parts of a program.
- Explain the role of scope in function behavior.
- Break larger problems into smaller functional components.
- Interpret and respond to exception messages while debugging.

Standards:

3A-AP-17	Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.
3A-AP-21	Evaluate and refine computational artifacts to make them more usable and accessible.
3A-AP-23	Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs.

Acceptable Evidence

Acceptable evidence of learning other than Performance Tasks:

Formative Assessment

- Warm-ups
- Check point assignments
- Read and Take Notes (RTN)

- Class work
- Homework
- Exit slips
- Keep a notebook
- Quizzes
- Programming Assignments
- Unit Quiz

Summative Assessment

- Unit Assessment

Suggested learning experiences and instruction

Learning Experiences:

- Use of projector to demonstrate new material.
- Use of Power Point slides to present concepts and terminology.

Instructional Strategies:

- Emphasize the use of algorithms in problem solving.
- Emphasize the importance of testing every function within a program.
- Complete teacher-generated self-review questions, multiple-choice, true/false, and short answer problems.

Differentiation:

- Work with students in small groups to intervene or enrich as needed
- Provide additional scaffolds for emerging learners.
- Allow on level learners to modify existing code and explain their changes.
- Provide opportunities for advanced learners to add features, variables, or additional functionality.

For Language Development:

- Create a student-made dictionary that includes terms, definitions, and examples organized by chapter.
- Make flash cards for terms and examples, then mix and match.
- Create study guides with clear definitions and vocabulary development opportunities.

For Advanced Learners:

- Meet in small groups with students who need enrichment to provide instruction and learning activities on the optional skills followed up by an alternative and more complex assignment.
- Excuse the students from standard practice and use appropriately challenging practice that requires students to apply the concepts in more rigorous ways to challenge their thinking and allow students to go deeper on the same skills.
- Provide more rigorous opportunities for application of skills including enrichment projects.

Central Bucks School District – Course of Study

Unit 6: Strings

Desired Results

Essential Questions:

- How can strings be treated as structured data rather than just plain text?
- How do indexing and slicing help programmers extract and transform information?
- What does it mean for strings to be immutable, and why does that matter?
- How can loops and methods support efficient string processing?

Enduring Understandings:

Students will understand that…

- Strings are sequences of characters that can be inspected, traversed, and manipulated systematically.
- Indexing and slicing allow programmers to access specific portions of text precisely.
- Immutability affects how strings are updated and used in programs.
- Built-in methods and loops support powerful text-processing solutions.

Knowledge:

Students will know…

- String indexing and slicing
- Immutability
- Traversing strings with for loops
- Membership testing with the in keyword
- Common string methods and formatting ideas

Skills:

Students will be able to…

- Access and manipulate characters and substrings using indices and slices.
- Use loops to examine and process text one character at a time.
- Apply string methods to clean, transform, or analyze text.
- Explain the implications of string immutability in program design.
- Build programs that solve text-based problems accurately and efficiently.

Standards:

3A-AP-13	Create prototypes that use algorithms to solve computational problems by leveraging prior student knowledge and personal interests.
3A-AP-17	Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.
3A-AP-21	Evaluate and refine computational artifacts to make them more usable and accessible.

Acceptable Evidence

Acceptable evidence of learning other than Performance Tasks:

Formative Assessment

- Warm-ups
- Check point assignments
- Read and Take Notes (RTN)

- Class work
- Homework
- Exit slips
- Keep a notebook
- Quizzes
- Programming Assignments
- Unit Quiz

Summative Assessment

- Unit Assessment

Suggested learning experiences and instruction

Learning Experiences:

- Use of projector to demonstrate new material.
- Use of Power Point slides to present concepts and terminology.

Instructional Strategies:

- Emphasize the use of algorithms in problem solving.
- Complete teacher-generated self-review questions, multiple-choice, true/false, and short answer problems.

Differentiation:

- Work with students in small groups to intervene or enrich as needed
- Provide additional scaffolds for emerging learners.
- Allow on level learners to modify existing code and explain their changes.
- Provide opportunities for advanced learners to add features, variables, or additional functionality.

For Language Development:

- Create a student-made dictionary that includes terms, definitions, and examples organized by chapter.
- Make flash cards for terms and examples, then mix and match.
- Create study guides with clear definitions and vocabulary development opportunities.

For Advanced Learners:

- Meet in small groups with students who need enrichment to provide instruction and learning activities on the optional skills followed up by an alternative and more complex assignment.
- Excuse the students from standard practice and use appropriately challenging practice that requires students to apply the concepts in more rigorous ways to challenge their thinking and allow students to go deeper on the same skills.
- Provide more rigorous opportunities for application of skills including enrichment projects.

Central Bucks School District – Course of Study

Unit 7: Data Structures

Desired Results

Essential Questions:

- How do data structures help programmers manage larger collections of information?
- When should a programmer use a tuple, list, 2D list, or dictionary?
- How do traversal, list methods, and comprehensions simplify data processing?
- How does a culminating project demonstrate planning, testing, debugging, and revision across multiple course concepts?

Enduring Understandings:

Students will understand that…

- Data structures organize information so programs can store, retrieve, and update data efficiently.
- Different data structures are useful for different tasks and design goals.
- Traversing and transforming collections is a foundational skill in programming.
- A complete computational artifact emerges through iterative development, testing, and refinement.

Knowledge:

Students will know…

- Tuples and lists
- List methods and list traversal
- 2D lists and nested iteration
- List comprehensions
- Packing, unpacking, and mutability concepts
- Dictionaries and key-value organization
- Equivalence vs. identity
- Project planning, debugging, testing, and revision practices

Skills:

Students will be able to…

- Create and modify lists and tuples appropriately.
- Traverse collections with loops to analyze or transform stored data.
- Use list methods and comprehensions to simplify code.
- Work with 2D lists and dictionaries to represent more complex information.
- Explain the difference between equivalence and identity in Python.
- Design, build, test, and refine a culminating Python project that integrates course concepts.

Standards:

3A-AP-14	Use lists to simplify solutions, generalizing computational problems instead of repeatedly using simple variables.
3A-AP-15	Justify the selection of specific control structures when tradeoffs involve implementation, readability, and program performance, and explain the benefits and drawbacks of choices made.
3A-AP-17	Decompose problems into smaller components through systematic analysis, using constructs such as procedures, modules, and/or objects.

3A-AP-21	Evaluate and refine computational artifacts to make them more usable and accessible.
3A-AP-23	Document design decisions using text, graphics, presentations, and/or demonstrations in the development of complex programs.
3A-DA-09	Translate between different bit representations of real-world phenomena, such as characters, numbers, and images.

Acceptable Evidence

Acceptable evidence of learning other than Performance Tasks:
Formative Assessment

- Warm-ups
- Check point assignments
- Read and Take Notes (RTN)
- Class work
- Homework
- Exit slips
- Keep a notebook
- Quizzes
- Programming Assignments
- Unit Quiz

Summative Assessment

- Unit Assessment

Suggested learning experiences and instruction

Learning Experiences:

- Use of projector to demonstrate new material.
- Use of Power Point slides to present concepts and terminology.

Instructional Strategies:

- Emphasize the use of algorithms in problem solving.
- Complete teacher-generated self-review questions, multiple-choice, true/false, and short answer problems.

Differentiation:

- Work with students in small groups to intervene or enrich as needed
- Provide additional scaffolds for emerging learners.
- Allow on level learners to modify existing code and explain their changes.
- Provide opportunities for advanced learners to add features, variables, or additional functionality.

For Language Development:

- Create a student-made dictionary that includes terms, definitions, and examples organized by chapter.
- Make flash cards for terms and examples, then mix and match.
- Create study guides with clear definitions and vocabulary development opportunities.

For Advanced Learners:

- Meet in small groups with students who need enrichment to provide instruction and learning activities on the optional skills followed up by an alternative and more complex assignment.
- Excuse the students from standard practice and use appropriately challenging practice that requires students to apply the concepts in more rigorous ways to challenge their thinking and allow students to go deeper on the same skills.
- Provide more rigorous opportunities for application of skills including enrichment projects.