**Computational Thinking with Python: Weekly Teaching Resources**

**📘 Week 1: Introduction to Computational Thinking & Python**

**Lesson Objectives:**

* Define computational thinking and its core components.
* Explain the Python programming environment.
* Write and run a basic Python program using print().

**Key Concepts:**

* Computational Thinking: Decomposition, Pattern Recognition, Abstraction, Algorithms
* Introduction to Python syntax
* Using print() for output

**Suggested Agenda:**

1. **Warm-up Activity (5–10 min):**
   * Ask students to list the steps to brush their teeth — discuss as an example of algorithmic thinking.
2. **Direct Instruction (15 min):**
   * Slide deck or whiteboard intro to computational thinking concepts.
   * Introduce Python, show basic syntax and setup.
3. **Demonstration (10 min):**
   * Show how to write and run print("Hello, world!") using the Python interpreter or IDE.
4. **Guided Practice (15–20 min):**
   * Students follow along to write and run small print() statements (name, hobby, favorite food).
5. **Independent Practice (15 min):**
   * Complete one or more assignments from the list.
6. **Exit Ticket (5 min):**
   * Ask students to write one thing they learned and one question they still have.

**Materials Needed:**

* Projector or whiteboard
* Student access to Python (IDLE, VS Code, or Replit)
* Printed or digital lesson outline/assignments

**Optional Enrichment:**

* Read short article by Jeannette Wing: "What is Computational Thinking?"
* Ask students to find a real-world task they could break into computational steps.

**📂 Week 2: Variables, Data Types, and Input/Output**

**Lesson Objectives:**

* Understand variables and basic data types in Python.
* Use input() and print() for user interaction.
* Convert between strings, integers, and floats.

**Key Concepts:**

* Variables and assignment
* Data types: int, float, str, bool
* Using input and print statements

**Suggested Agenda:**

1. **Warm-up (5 min):**
   * Display a few sample values and ask students to guess their data types.
2. **Direct Instruction (15 min):**
   * Explain and show examples of different data types.
   * Demonstrate input() and type conversion.
3. **Demonstration (10 min):**
   * Show how to store and manipulate user input.
4. **Guided Practice (15 min):**
   * Students write programs to get and print user data.
5. **Independent Practice (15 min):**
   * Complete assignment problems for reinforcement.
6. **Wrap-up (5 min):**
   * Recap how Python handles input and types.

**Materials Needed:**

* Sample data type table
* Access to code editor or notebook

**Optional Enrichment:**

* Mini-research task: Find and explain a new built-in data type in Python.

**📈 Week 3: Operators and Expressions**

**Lesson Objectives:**

* Use arithmetic and logical operators in Python expressions.
* Understand operator precedence.
* Write expressions to solve real-world problems.

**Key Concepts:**

* Arithmetic operators: +, -, \*, /, %, //, \*\*
* Logical operators: and, or, not
* Comparison operators: >, <, ==, !=, >=, <=

**Suggested Agenda:**

1. **Warm-up (5 min):**
   * Solve a math problem using order of operations.
2. **Direct Instruction (15 min):**
   * Introduce each operator type with examples.
3. **Demo (10 min):**
   * Evaluate and print sample expressions.
4. **Guided Practice (15 min):**
   * Students write expressions and test outcomes.
5. **Independent Practice (15 min):**
   * Students write simple programs using operators.
6. **Exit Ticket:**
   * Provide an expression and ask students to predict its result.

**Materials Needed:**

* Operator cheat sheet
* Worksheets with fill-in-the-blank expressions

**Optional Enrichment:**

* Introduce modulus operator in real-world contexts (e.g., clock math).

**🧮 Week 4: Conditional Statements**

**Lesson Objectives:**

* Use if, elif, and else statements for decision-making.
* Compare values using relational and logical operators.
* Write simple branching programs.

**Key Concepts:**

* Conditional logic
* Branching with multiple conditions
* Nesting and indentation rules

**Suggested Agenda:**

1. **Warm-up (5 min):**
   * Ask: “What decisions did you make before school today?”
2. **Direct Instruction (15 min):**
   * Demonstrate conditional syntax and logic.
3. **Demo (10 min):**
   * Walk through a grading or number-classification example.
4. **Guided Practice (15 min):**
   * Have students code a branching program with peer support.
5. **Independent Practice (15 min):**
   * Use prompts to write and test condition-based programs.
6. **Wrap-up (5 min):**
   * Challenge: Write one real-world decision as an if statement.

**Materials Needed:**

* Whiteboard flowchart
* Practice condition cards

**Optional Enrichment:**

* Add nested if conditions for more complex logic.

**🔁 Week 5: Loops - Part 1 (While Loops)**

**Lesson Objectives:**

* Understand how and when to use while loops.
* Implement loop control using break and continue.
* Write programs that repeat based on conditions.

**Key Concepts:**

* Looping and iteration
* while loop syntax
* Infinite loops and control flow

**Suggested Agenda:**

1. **Warm-up (5 min):**
   * Predict the output of a basic while loop.
2. **Direct Instruction (15 min):**
   * Explain while loop logic and show structure.
3. **Demo (10 min):**
   * Show a number guessing game or countdown timer.
4. **Guided Practice (15 min):**
   * Write a simple while loop together.
5. **Independent Practice (15 min):**
   * Students create a loop-based program.
6. **Exit Ticket:**
   * Describe a scenario where a while loop is useful.

**Materials Needed:**

* Sample loop diagrams
* Example problems for modeling

**Optional Enrichment:**

* Add complexity with nested loops or timers.

**🔢 Week 6: Loops - Part 2 (For Loops)**

**Lesson Objectives:**

* Use for loops to iterate through sequences.
* Loop through ranges, strings, and lists.
* Identify cases best suited for for loops.

**Key Concepts:**

* for loop structure
* range() function
* Sequence iteration

**Suggested Agenda:**

1. **Warm-up (5 min):**
   * Trace the output of a simple for loop.
2. **Direct Instruction (15 min):**
   * Compare for vs. while and introduce use cases.
3. **Demo (10 min):**
   * Loop through a string and a list.
4. **Guided Practice (15 min):**
   * Students write a loop to print even numbers.
5. **Independent Practice (15 min):**
   * Write a loop to process user input data.
6. **Exit Ticket:**
   * When would you use a for loop instead of a while loop?

**Materials Needed:**

* Code snippets
* Loop table templates

**Optional Enrichment:**

* Introduce nested loops for advanced students.

(Continuing with Weeks 7–16 next...)

**Computational Thinking with Python: Weekly Teaching Resources**

**📘 Week 1: Introduction to Computational Thinking & Python**

**Lesson Objectives:**

* Define computational thinking and its core components.
* Explain the Python programming environment.
* Write and run a basic Python program using print().

**Key Concepts:**

* Computational Thinking: Decomposition, Pattern Recognition, Abstraction, Algorithms
* Introduction to Python syntax
* Using print() for output

**Suggested Agenda:**

1. Warm-up Activity: List steps of a simple task.
2. Direct Instruction: Computational thinking and basic syntax.
3. Demonstration: Write and run a print() statement.
4. Guided Practice: Print name, hobby, favorite food.
5. Independent Practice: Complete short assignment.
6. Exit Ticket: Share one takeaway and one question.

**Materials Needed:**

* Projector or whiteboard, IDE access

**📂 Week 2: Variables, Data Types, and Input/Output**

**Lesson Objectives:**

* Use and identify variables and data types.
* Accept user input with input().
* Convert input to the correct data type.

**Key Concepts:**

* Variables, strings, integers, floats, booleans
* input() and print()

**Suggested Agenda:**

1. Warm-up: Guess the data type.
2. Instruction: Explain variables and data types.
3. Demo: Use input and conversion.
4. Guided Practice: Input user data.
5. Independent Practice: Write basic input/output programs.
6. Wrap-up: Recap with quick quiz or Q&A.

**📈 Week 3: Operators and Expressions**

**Lesson Objectives:**

* Use arithmetic, comparison, and logical operators.
* Apply operator precedence.

**Key Concepts:**

* +, -, \*, /, %, \*\*, //
* >, <, ==, !=, and, or, not

**Suggested Agenda:**

1. Warm-up: Math puzzle with precedence.
2. Instruction: Overview of operator types.
3. Demo: Show examples of each.
4. Guided Practice: Evaluate expressions.
5. Independent Practice: Create and test expressions.
6. Exit Ticket: Solve one expression with multiple operators.

**🧮 Week 4: Conditional Statements**

**Lesson Objectives:**

* Use if, elif, else for decision-making.
* Build logic-based flow in code.

**Key Concepts:**

* Conditional syntax
* Boolean logic
* Nesting conditions

**Suggested Agenda:**

1. Warm-up: Discuss a real-world decision process.
2. Instruction: Explain branching and syntax.
3. Demo: Write a grading program.
4. Guided Practice: Check number sign (positive/negative/zero).
5. Independent Practice: Write personalized branching program.
6. Wrap-up: Translate a real decision into code.

**🔁 Week 5: Loops - Part 1 (While Loops)**

**Lesson Objectives:**

* Understand and use while loops.
* Control repetition with conditions.

**Key Concepts:**

* Loop initialization and updating
* Avoiding infinite loops
* Loop control: break, continue

**Suggested Agenda:**

1. Warm-up: Predict the output of a loop.
2. Instruction: Introduce while loop structure.
3. Demo: Countdown or guessing game.
4. Guided Practice: Simple while loop programs.
5. Independent Practice: Write number sum or validator.
6. Exit Ticket: Identify loop use case.

**🔢 Week 6: Loops - Part 2 (For Loops)**

**Lesson Objectives:**

* Use for loops to iterate over sequences.
* Compare for and while loops.

**Key Concepts:**

* for loops and range()
* Strings and list iteration

**Suggested Agenda:**

1. Warm-up: Trace a for loop.
2. Instruction: for loop syntax and uses.
3. Demo: Loop through string and number range.
4. Guided Practice: Create basic for loop.
5. Independent Practice: Create even number printer.
6. Exit Ticket: Share difference between for and while.

**🧾 Week 7: Functions and Modular Programming**

**Lesson Objectives:**

* Define and call functions.
* Use parameters and return values.

**Key Concepts:**

* def statements
* Parameters, return values
* Code reuse/modularity

**Suggested Agenda:**

1. Warm-up: Why use functions?
2. Instruction: Syntax and structure of functions.
3. Demo: Create a function with arguments.
4. Guided Practice: Create temperature converter.
5. Independent Practice: Build simple function-driven programs.
6. Exit Ticket: Write a function header and describe it.

**📋 Week 8: Lists and Basic Data Structures**

**Lesson Objectives:**

* Create and manipulate lists.
* Use list methods to modify content.

**Key Concepts:**

* Indexing, slicing
* append(), remove(), len()

**Suggested Agenda:**

1. Warm-up: List your morning routine.
2. Instruction: Explain list syntax and behavior.
3. Demo: Add/remove list items.
4. Guided Practice: Create shopping list manager.
5. Independent Practice: Analyze and transform lists.
6. Wrap-up: Discuss real-world list examples.

**📊 Week 9: Problem Solving with Lists**

**Lesson Objectives:**

* Use loops and lists for problem-solving.
* Implement list comprehensions.

**Key Concepts:**

* Iterating over lists
* Searching, filtering
* List comprehensions

**Suggested Agenda:**

1. Warm-up: What data could you store in a list?
2. Instruction: Search, sort, and comprehension intro.
3. Demo: Count elements and generate new lists.
4. Guided Practice: Create list of squares.
5. Independent Practice: Search/filter examples.
6. Exit Ticket: Describe list comprehension.

**✉️ Week 10: Strings and Text Processing**

**Lesson Objectives:**

* Use string methods for analysis and transformation.
* Format and parse strings.

**Key Concepts:**

* .lower(), .upper(), .replace(), .count()
* split(), join()

**Suggested Agenda:**

1. Warm-up: Count letters in your name.
2. Instruction: Common string methods.
3. Demo: String case change, format name.
4. Guided Practice: Count vowels or words.
5. Independent Practice: Reformat or encode strings.
6. Exit Ticket: Share one string method learned.

**📖 Week 11: Introduction to Dictionaries**

**Lesson Objectives:**

* Store and retrieve data using key-value pairs.
* Apply dictionary methods for common tasks.

**Key Concepts:**

* Dictionaries and their syntax
* .get(), .keys(), .values()

**Suggested Agenda:**

1. Warm-up: Match names to phone numbers.
2. Instruction: Define and use dictionaries.
3. Demo: Add and retrieve dictionary entries.
4. Guided Practice: Phone book simulation.
5. Independent Practice: Word frequency counter.
6. Exit Ticket: Share one use for a dictionary.

**🧠 Week 12: Algorithms and Algorithmic Thinking**

**Lesson Objectives:**

* Understand and implement basic search algorithms.
* Write pseudocode for simple algorithms.

**Key Concepts:**

* Linear search, binary search (conceptual)
* Sorting basics
* Efficiency discussion

**Suggested Agenda:**

1. Warm-up: Find a number in a list by hand.
2. Instruction: Linear search explained.
3. Demo: Write pseudocode for search/sort.
4. Guided Practice: Trace and code linear search.
5. Independent Practice: Sort with built-in functions.
6. Wrap-up: Efficiency and tradeoffs.

**🐞 Week 13: Debugging and Testing**

**Lesson Objectives:**

* Identify and fix common errors in Python.
* Write basic test cases.

**Key Concepts:**

* Syntax, runtime, logic errors
* try/except, assert for testing

**Suggested Agenda:**

1. Warm-up: What kinds of bugs have you encountered?
2. Instruction: Error types and debugging strategies.
3. Demo: Debug a sample script.
4. Guided Practice: Find bugs in pairs.
5. Independent Practice: Write tests for known code.
6. Exit Ticket: Name a test you would write.

**🔁 Week 14: Introduction to Recursion**

**Lesson Objectives:**

* Understand recursive function structure.
* Implement basic recursive algorithms.

**Key Concepts:**

* Base case, recursive step
* Factorial, Fibonacci

**Suggested Agenda:**

1. Warm-up: What is repetition in nature or math?
2. Instruction: Explain recursive logic.
3. Demo: Factorial and Fibonacci.
4. Guided Practice: Trace recursive calls.
5. Independent Practice: Build own recursive function.
6. Wrap-up: When to use recursion.

**🛠️ Week 15: Final Project Work**

**Lesson Objectives:**

* Plan, design, and begin coding a final project.
* Apply skills learned to solve a real problem.

**Key Concepts:**

* Project planning
* Input, process, output structure
* Collaboration

**Suggested Agenda:**

1. Review: Final project guidelines.
2. Activity: Brainstorm and pitch ideas.
3. Practice: Write pseudocode.
4. Develop: Begin implementing first features.
5. Peer Review: Share progress and feedback.

**🎓 Week 16: Project Presentations and Wrap-Up**

**Lesson Objectives:**

* Present and explain final projects.
* Reflect on computational thinking journey.

**Key Concepts:**

* Communication of technical ideas
* Project reflection
* Growth and next steps

**Suggested Agenda:**

1. Present: Students showcase projects.
2. Reflect: What was learned? Challenges?
3. Celebrate: Acknowledge effort and achievement.
4. Close: Feedback, course wrap-up, future pathways.

✅ All 16 weeks now include detailed teaching resources. Let me know if you'd like them formatted as slides, handouts, or a teacher’s guide!