Professional Development Session: Bridging Classroom Coding with Workforce Skills

Duration: 90 minutes

Objective: Provide CollegeNow CS instructors with the essential knowledge and tools to incorporate key workforce skills into their coding curriculum, focusing on unit testing, debugging techniques and version control with Git.

Overview of Workforce Skills

The significance of unit testing, version control, and debugging in the tech industry:

- **Unit Testing**:
    - **Ensures Code Quality**: Unit tests help verify that individual pieces of code work as intended, reducing the risk of bugs in later stages of development.
    - **Facilitates Refactoring**: With a robust set of unit tests, developers can refactor code with confidence, knowing that they will be alerted to any issues introduced.
    - **Encourages Better Design**: Writing unit tests forces developers to think about the code design and modularity, leading to cleaner, more maintainable code.
    - **Industry Standard**: Practically all professional software development teams use unit testing to maintain code quality and reliability.

- **Version Control with Git**:
    - **Tracks Changes**: Git allows developers to keep a detailed history of code changes, making it easier to track and revert to previous versions if necessary.
    - **Facilitates Collaboration**: Multiple developers can work on the same project simultaneously, with Git managing changes and merging contributions from different team members.
    - **Improves Workflow**: Branching and merging in Git support different workflows, enabling features like parallel development, code reviews, and continuous integration.
    - **Industry Standard**: Knowledge of Git is a fundamental skill expected in nearly all software development jobs, as it is used for version control in most projects.

- **Debugging Techniques**:
    - **Essential for Problem-Solving**: Debugging is a critical skill for identifying and resolving errors in code, ensuring software functions as intended.
    - **Enhances Code Understanding**: Debugging helps developers understand the inner workings of their code and the systems it interacts with. This aspect is especially useful in coding education as debuggers allow us to analyze memory models of data structures and step-by-step tracing of logic
    - **Improves Efficiency**: Effective debugging techniques can significantly reduce the time and effort required to fix issues, leading to more efficient development processes.

- **Industry Standard**: Proficiency with debugging tools and techniques is expected in professional settings, as it directly impacts the quality and reliability of the final product.

Integration into Curriculum

Benefits of integrating these skills into high school coding classes:

- **Preparation for Higher Education**: Introducing unit testing, version control, and debugging in high school prepares students for more advanced computer science courses in college, where these skills are often assumed knowledge. Below you can see the guidelines laid out by the SUNY Seamless Transfer initiative for the first two CS courses. Read more here

https://www.suny.edu/attend/get-started/transfer-students/suny-transfer-paths/pdf/transferSUNY_Computer_Science.pdf

Computer Science I

This course covers the fundamentals of computer problem solving and programming. Topics include: program development process, differences between the object-oriented, structured, and functional programming methodologies, phases of language translation (compiling, interpreting, linking, executing), and error conditions associated with each phase, primitive data types, memory representation, variables, expressions, assignment, fundamental programming constructs (sequence, selection, iteration), algorithms for solving simple problems, tracing execution, subprograms/functions/methods, parameter passing, secure coding techniques (criteria for selection of a specific type and use, input data validation), and professional behavior in response to ethical issues inherent in computing.

Computer Science II

This course covers the fundamentals of algorithms and object oriented software development. Topics include: modern IDE for software development, primitive and reference data types, encapsulation, information hiding, selection, iteration, functions/methods, parameters, recursion, exception handling, generic linear data structures (arrays, records/structs) and maps, file types, file I/O, simple GUIs with event handling, programming to an interface, lambda expressions, semantics of inheritance and use of polymorphism, relation with subtyping, search (sequential, binary), select (min, max), and sort (bubble, insertion, selection) algorithms, complexity notation, documentation using standard tools, program testing (unit testing) and debugging, reasoning about control flow in a program, and societal impacts related to computing and software.

- **Career Readiness**: Students gain practical skills that are directly applicable in the tech industry, making them more attractive to potential employers and better prepared for internships and job opportunities.
- **Improved Problem-Solving Skills**: Learning debugging techniques enhances students' problem-solving abilities, fostering a deeper understanding of programming logic and error resolution.
- **Promotes Best Practices**: Teaching these skills instills best practices early on, encouraging students to write clean, maintainable code and work collaboratively from the start of their coding journey.

- **Enhanced Project Management**: Understanding version control helps students manage their projects more effectively, keeping their work organized and enabling efficient collaboration on group projects.
- **Increased Confidence and Independence**: Students equipped with these skills can troubleshoot and solve problems independently, leading to increased confidence in their coding abilities.
- **Alignment with Industry Standards**: By aligning the curriculum with industry standards, educators ensure that students are learning relevant, up-to-date practices, bridging the gap between education and the professional world.

Code Walk Through – Hands On Activity: In this activity we will use Python to explore two different types of function testing – doctests and unit tests. To follow along you will need some version of Python 3 installed. The demo will use Python 3.10.11 and VS Code with the Python extension v2024.12.3. These exact versions are not mandatory.

Get the repository here: https://github.com/kwhitener/TC3-CollegeNow-Summer-2024.git

- To clone the repository, you will need to have **git** installed
- Open your terminal of choice and navigate to your workspace.
- **Execute: git clone https://github.com/kwhitener/TC3-CollegeNow-Summer-2024.git**
- Navigate into the repository

Deterministic vs non-deterministic functions.

The distinction between deterministic and non-deterministic functions is fundamental in computer science and mathematics, especially in the context of algorithms and computational complexity. Here are the key differences:

Deterministic Functions

1. **Predictability**: A deterministic function always produces the same output given the same input. The behavior is entirely predictable and repeatable.
2. **Single Path Execution**: There is only one path of execution. For a given input, the function follows a specific sequence of steps to reach the output.
3. **Examples**:
    - Mathematical functions like `f(x)=2x+3` are deterministic. For any given value of x, the output is always the same.
    - Sorting algorithms like Merge Sort or Bubble Sort are deterministic. Given the same unsorted list, they will always produce the same sorted list.
4. **Debugging and Testing**: Debugging and testing are straightforward since the function behaves consistently. Any observed behavior can be reproduced and analyzed. Deterministic functions in Python can be tested using doctests. Doctests not only serve as behavioral sanity checks that can be re-executed when unit logic is refactored, they also serve as great documentation of function usage

Great practical introduction: https://realpython.com/python-doctest/
Official documentation: https://docs.python.org/3/library/doctest.html

Non-Deterministic Functions

1. **Unpredictability**: A non-deterministic function may produce different outputs given the same input. The behavior can vary between different executions.
2. **Multiple Path Execution**: There are multiple potential paths of execution for the same input. This could be due to inherent randomness, concurrency issues, or external factors influencing the function.
3. **Examples**:
    - A function that generates random numbers, like `random.randint(1, 10)`, is non-deterministic because it can produce any number in the specified range each time it is called.
    - Concurrent or parallel algorithms where the outcome may depend on the scheduling of threads or processes, such as certain implementations of quicksort or distributed systems operations.
4. **Debugging and Testing**: Debugging and testing are more challenging due to the variability in behavior. Reproducing specific issues can be difficult because the function might not exhibit the same behavior consistently. We need to utilize techniques that allow logical structures in the test, the classic “write a program to test the program” paradox.

    We will use Python’s **unittest** module: https://docs.python.org/3/library/unittest.html