Universal Design for Learning Guidelines + Computer Science / Computational Thinking

	Multiple Means of **Engagement**	Multiple Means of **Representation**	Multiple Means of **Action & Expression**
	Affective Networks The "WHY" of learning	Recognition Networks The "WHAT" of learning	Strategic Networks The "HOW" of learning
Access	Provide options for **Recruiting Interest** ● Give students choices (choose project, software, topic) ● Allow students to make projects relevant to culture and age ● Minimize possible common "pitfalls" for both computing and content ● Allow for differences in pacing and length of work sessions ● Provide options to increase or decrease sensory stimulation (for example listening to music with headphones or using noise cancelling headphones) ● Allow for differences in pacing and length of work sessions	Provide options for **Perception** ● Model computing using physical representations as well as through an interactive whiteboard, videos ● Give access to modeled code while students work independently ● Provide access to video tutorials of computing tasks ● Select coding apps and websites that allow the students to adjust visual settings (such as font size & contrast) and that are compatible with screen readers.	Provide options for **Physical Action** ● Provide teacher's codes as templates ● Include CS Unplugged activities that show physical relationship of abstract computing concepts ● Use assistive technology including larger/smaller mice, touch-screen devices ● Select coding apps and websites that allow coding with keyboard shortcuts in addition to dragging & dropping with a mouse

	Multiple Means of **Engagement**	Multiple Means of **Representation**	Multiple Means of **Action & Expression**
	Affective Networks The "WHY" of learning	Recognition Networks The "WHAT" of learning	Strategic Networks The "HOW" of learning
Build	Provide options for **Sustaining Effort & Persistence** • Remind students of both computing and content goals • Provide support or extensions for students to keep engaged • Teach and encourage peer collaboration by sharing products • Utilize pair programming and group work with clearly defined roles • Discuss the integral role of perseverance and problem solving in computer science • Recognize students for demonstrating perseverance and problem solving in the classroom	Provide options for **Language & Symbols** • Teach and review content specific vocabulary • Teach and review computing vocabulary (e.g., code, animations, computing, algorithm) • Post anchor charts and provide reference sheets with images of blocks or with common syntax when using text	Provide options for **Expression & Communication** • Give options of unplugged activities and computing software and materials (e.g., Pseudocode, Scratch, code.org, Alice) • Give opportunities to practice computing skills and content through projects that build prior lessons • Provide sentence starters or checklists for communicating in order to collaborate, give feedback, and explain work • Create physical manipulatives of commands, blocks or lines of code • Provide options that include starter code

Israel, M., Lash, T., Ray, M. (2017). Universal Design for Learning within Computer Science Education. *Creative Technology Research Lab.* University of Florida.

	Multiple Means of **Engagement**	Multiple Means of **Representation**	Multiple Means of **Action & Expression**
	Affective Networks The "WHY" of learning	Recognition Networks The "WHAT" of learning	Strategic Networks The "HOW" of learning
Internalize	Provide options for **Self Regulation** ● Communicate clear expectations for computing tasks, collaboration, and help seeking ● Develop ways for students to self-assess and reflect on own projects and those of others ● Use assessment rubrics that evaluate both content and process ● Break-up coding activities with opportunities for reflection such as turn and talks or written questions ● Acknowledge difficulty and frustration. Model different strategies for dealing with frustration appropriately	Provide options for **Comprehension** ● Activate background knowledge by making computing tasks interesting and culturally relevant ● State lesson content/ computing goals ● Encourage students to ask questions as comprehension checkpoints ● Use relevant analogies and make cross-curricular connections explicit (for example comparing iterative product development to the writing process) ● Provide graphic organizers for students to "translate" programs into pseudocode	Provide options for **Executive Functions** ● Guide students to set goals for long-term projects ● Record students' progress (have planned checkpoints during lessons for understanding and progress for computing skills and content) ● Provide exemplars of completed products ● Embed prompts to stop and plan, test, or debug throughout a lesson or project. ● Provide graphic organizers to facilitate planning, goal-setting, and debugging ● Provide explicit instruction on skills such as asking for help, providing feedback, and using problem solving techniques ● Demonstrate debugging with think-alouds