# A Macro Look at Microworlds

## Teachers Designing Small Worlds Where Big Ideas Live

*A work in progress...*


**Gary S. Stager, PhD**

*CEO: Constructing Modern Knowledge*

gary@stager.org

professorgarystager.com


May 2026

# AI Generated Executive Summary

*A Macro Look at Microworlds* — with practitioner handouts for Snap!, Finch, and Turtle Art

---

**Overview.** This document pairs a conceptual paper on educational microworlds with a parallel-tradition appendix and three hands-on tutorials. Together, the materials make both the case for microworlds as a pedagogical practice and the case that any classroom teacher — not only a software developer — can design one.

**A tradition with deep roots.** The microworld idea was first articulated in the 1970s at MIT's AI Laboratory and popularized through Seymour Papert's *Mindstorms*. It rests on a constructionist foundation drawn from Piaget: knowledge is built rather than transmitted, and a computer is most powerful when used as an *object-to-think-with*. Lawler offered the design criterion of *neat phenomena* — material inherently interesting to probe; Feurzeig insisted that microworlds, like real worlds, "do not teach" but invite exploration; Dan and Molly Watt showed how teacher-made microworlds let young children act as mathematicians, poets, and artists rather than merely study those subjects. Stager extends this lineage by reading microworlds through the Reggio Emilia tradition: the prepared environment as a "third teacher," and the classroom teacher as curator of a computational atelier.

**Why microworlds matter.** Microworlds are, in Stager's phrase, "one of education's most powerful and underutilized ideas." They reframe educational technology away from delivery and toward construction. They generate intrinsic motivation by surrounding students with phenomena worth investigating; they invite personal appropriation by letting learners make ideas their own; they make debugging — and therefore thinking about thinking — a routine habit; and they carry powerful ideas (variable separation through POLYSPI, Newtonian motion through Dynaturtle, language as command through word-action worlds) into ages and classrooms that conventional curricula reach much later or not at all. They make Bruner's claim operational: that *any subject can be taught effectively in some intellectually honest form to any child at any stage of development*. And they put teachers, not vendors, in the driver's seat — because only a teacher who knows particular learners can decide which complexities to hide, which to reveal, and when. Stager is candid that this places real demands on teachers: left alone in an open environment, many children thrash; the microworld's power is unlocked by skilled adult guidance and by an iterative, observational stance toward children's thinking. He is equally candid about the field's persistent failure mode — Papert's "domestication," the slow conversion of open exploratory environments into scripted exercises — which the paper treats as the practice's central contemporary challenge.

**Vocabulary and apparatus.** The paper supplies the working vocabulary teachers need to design microworlds and to recognize them in the wild: *primitives* (the blocks built into a system) and *pseudo-primitives* (teacher-defined blocks that become the raw material of a microworld), together with the recursive observation that environments like Logo, Scratch, and Snap! are themselves microworlds within which more constrained or enhanced ones can be built.

**Appendices.** Two appendices extend the argument. **Appendix A** treats Parsons Problems — scrambled-code puzzles developed independently in CS-education research — as evidence that a separate research community arrived at compatible conclusions, with a useful caveat about not mistaking them for assessments of broader programming ability. **Appendix B** provides three step-by-step tutorials that turn the theory into Monday-morning practice: building a custom palette in Snap!, programming a Finch robot as a floor turtle, and creating a shareable microworld in Turtle Art.

**Takeaway.** This working document is both a manifesto and a starter kit. It supplies the theoretical grounding teachers need to defend microworld pedagogy in their schools and the concrete tools to begin building microworlds this week — making the case that the most powerful educational technology decisions are made not by vendors but by teachers who know their students well enough to design small worlds in which big ideas can take root.

# A Macro Look at Microworlds

*Teachers Designing Small Worlds Where Big Ideas Live*

Gary S. Stager, Ph.D.
Founder and CEO: Constructing Modern Knowledge
gary@stager.org

## Abstract

Microworlds are teacher-crafted, purposefully bounded learning environments (often built with tools like Logo/Scratch/Snap!/MakeCode) where students learn by exploring "neat" phenomena, making things, and debugging their ideas, rather than by following step-by-step instruction. Grounded in constructionism, microworlds act as "objects-to-think-with" that connect new concepts to students' lived experience and invite personal ownership through the construction of open-ended projects. In the classroom, the pedagogical design challenge is to balance constraint with freedom: simplify enough to help learners get started, make the system's rules and state visible so students can trace cause and effect, and keep the environment extensible so they can build new tools as their understanding grows. The paper emphasizes that microworlds succeed in a social setting, where students collaborate and a skilled teacher guides, questions, and refines the environment, so exploration becomes equitable, meaningful, and cumulative.

Following the theoretical and historical basis for microworlds, there are appendices exploring Parsons Problems, as well as instructions for designing microworlds in Snap! and Turtle Art. There is also a tutorial illustrating how a microworld might be designed to allow young children to program a Finch robot.

## Understanding Microworlds: Creating Powerful Learning Environments for Students

In the landscape of educational technology, few concepts have proven as enduring and transformative as the idea of microworlds. First articulated in the 1970s at MIT's Artificial Intelligence Laboratory and popularized through Seymour Papert's landmark book *Mindstorms*, microworlds represent both a pedagogical philosophy and a practical approach to creating rich learning environments. I have long described microworlds as "a form of scaffolding that gives flight to a student's thinking and adds efficiency to a project" (Stager, 2025). Microworlds reflect a teacher's understanding and respect for their students by placing teachers, rather than software developers, in the driver's seat. This stance makes the power inherent in open-ended software environments accessible to a diverse universe of students.

## What is a Microworld?

At its core, a microworld is a simplified, self-contained environment designed for exploration and learning. As Dan Watt explains, microworlds are "incubators of knowledge" where children can acquire powerful ideas by carrying out activities within them (Watt, 1985). Unlike traditional computer-assisted instruction that leads learners through predetermined sequences, microworlds provide open-ended spaces for discovery.

Wally Feurzeig, another pioneer in this field, emphasizes what microworlds are not: "Logo microworlds do not teach. Like real worlds, they do not give away their secrets or explain themselves to passers by—they simply exist and behave. They can be probed, their behavior can be explored, their structure understood, their underlying operation discovered" (Feurzeig). This fundamental difference separates microworlds from the vast majority of educational software, which remains focused on delivering instruction rather than enabling exploration.

The concept emerged from an unlikely source: robotics research. Papert notes that "the microworld idea was used for robots before it was used for children" (Papert, 2002). In early artificial intelligence work at MIT, researchers created simplified "Blocks Worlds" where robots could manipulate objects without dealing with the full complexity of the real environment. This same principle—creating bounded but rich spaces for learning—transferred powerfully to educational settings.

Robert (Bob) Lawler, a key figure in microworld development, offers specific criteria for effective design. A microworld should contain "neat phenomena—phenomena that are inherently interesting to observe and interact with" (Lawler, 1982). Beyond mere entertainment, these phenomena must embody powerful ideas that are simple enough to grasp, general enough to apply broadly, immediately useful to learners, and "syntonic"—closely related to learners' existing mental models.

The most famous example remains Logo's turtle graphics, where students command a screen turtle to draw geometric shapes. This microworld was designed as "a multi-tactile microworld for Kindergarten students" that "allows students to explore, through the use of robotic toys, drawing tablets, such as the Koala pad, Logo, and other sensory materials, the form and nature of the letters which they are learning."

## The Theoretical Foundation: Rooted in Constructionism

Microworlds rest on a constructionist foundation, drawing heavily from Jean Piaget's work on how children construct knowledge. As Lawler explains, this represents "constructivism, the view that the mind incorporates a natural growth of knowledge and that the mind's structure and organization are shaped by interactions among the mind's parts" (Lawler, 1982). This contrasts sharply with instructionist models where teachers transmit ready-made knowledge to passive recipients.

The concept of transitional objects plays a crucial role. Just as young children use physical objects like blocks to understand abstract concepts, microworlds provide computational transitional objects. Papert describes them as "a transitional object between the ones that you can touch and push (like

tables and wooden blocks) and the kind of objects that you know in science, in philosophy, and in mathematics" (Papert, 1987).

Microworlds also embody what Papert calls "mathetics"—principles that facilitate learning, as distinct from heuristics for problem-solving. Two key mathetic principles guide microworld design: "relate what is new and to be learned to something you already know" and "take what is new and make it your own: Make something new with it, play with it, build with it" (Papert, 1980).

Wally Feurzeig explains an essential distinction: unlike computer-assisted instruction systems, microworlds "do not teach. Like real worlds, they do not give away their secrets or explain themselves to passers by—they simply exist and behave. They can be probed, their behavior can be explored, their structure understood, their underlying operation discovered" (Feurzeig). This fundamental difference separates microworlds from the vast majority of educational software.

It's crucial to understand, however, that microworlds are not meant to replace teaching. Feurzeig warns against a "bizarre distortion" of the Logo philosophy—the notion "that exposing students to a semantically rich microworld will spontaneously generate discovery and invention, promote the development of general thinking skills (whatever these are), and engender problem-solving skills that transfer directly to other task domains" (Feurzeig). This is a misunderstanding. As Feurzeig emphasizes, "while CAI technology is often directed at teacherless teaching, Logo technology has always been committed to the social milieu of human interactions among children, teachers and computers" (Feurzeig).

The reality is that "without the aid of a teacher, many children do not learn in a Logo microworld. They are not able to set their own goals, to find effective methods of thinking about problems, or to acquire the skills of exploration, conjecture and inference. Left to themselves, they thrash about. They haven't learned how to function in an open learning environment" (Feurzeig). This acknowledgment doesn't diminish the power of microworlds—it simply clarifies that they work best when "skilled teachers overcome these deficits by providing the guidance and support that make microworld experiences productive for their students" (Feurzeig).

## The Computational Atelier: Microworlds and the Reggio Emilia Approach

One of the most powerful frameworks for understanding microworlds comes from an unlikely source: the Reggio Emilia approach to early childhood education developed in northern Italy. This connection, which I emphasize throughout my workshop work, reveals how microworlds embody principles that progressive educators have long valued, making computational thinking accessible to teachers who might otherwise see it as foreign to their practice.

### The Environment as Third Teacher

In Reggio Emilia pedagogy, the physical environment is considered "the third teacher"—as important as the adult educators and the children themselves. The arrangement of materials, the quality of light, the accessibility of tools—all communicate values, suggest possibilities, and shape

learning. As I note in my workshop description, microworlds can function as computational ateliers where “teacher-prepared computational materials and objects-to-think with” serve this same role (Stager, 2025).

When a teacher creates a word-action microworld for young readers, carefully selecting vocabulary that reflects the child’s lived experience and interests, they’re preparing the environment just as deliberately as a Reggio teacher arranges provocations in the atelier. The Farm microworld created for a four-year-old in a farming community, the BEACH microworld for coastal children, the city vehicles microworld for an urban child—each represents the same thoughtful curation of materials based on intimate knowledge of particular learners that characterizes Reggio practice.

The microworld itself becomes the third teacher. It doesn’t instruct or evaluate—rather, as Feurzeig notes, it “simply exists and behaves” (Feurzeig), inviting exploration and responding consistently to student actions. Like a well-prepared Reggio environment, it communicates respect for children’s capability, offers genuine choices, and supports multiple ways of engaging with powerful ideas.

## Deliberate Selection of Materials

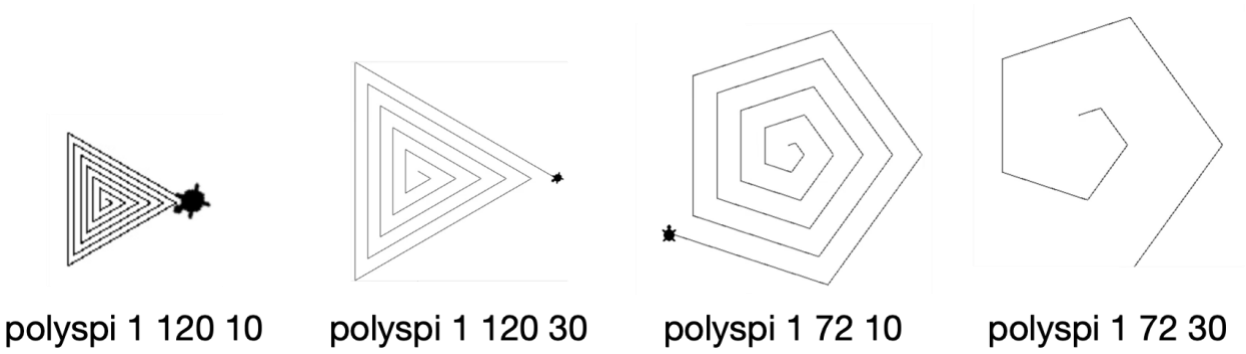
Reggio educators are famous for their careful selection of materials—not just any blocks, but blocks of particular woods with particular grains; not random colors but carefully considered palettes; not arbitrary tools but implements chosen to invite specific kinds of investigation. The materials themselves carry pedagogical intent.

Microworlds require the same thoughtful curation. When teachers create simplified Logo commands for young children—“F 10” instead of “FORWARD 100”—they’re making the same kind of deliberate choice about materials that a Reggio teacher makes in selecting wide-tipped markers instead of fine-point pens for toddlers. The goal isn’t to limit possibility but to make engagement more accessible, to remove barriers that might prevent exploration.

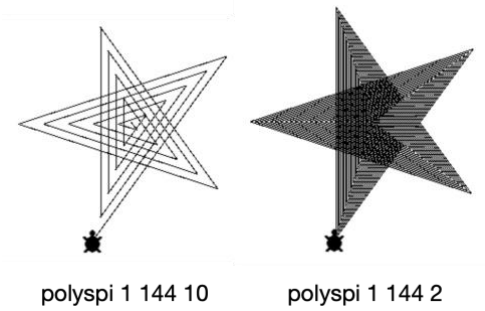
As I emphasize in my work, this is “a form of scaffolding that gives flight to a student’s thinking and adds efficiency to a project” (Stager, 2025). Just as clay of the right consistency invites manipulation while clay that’s too hard or too wet frustrates, computational materials must have the right “grain” for learners at different developmental stages. The teacher creating a microworld makes choices about which procedures to provide, which to leave for students to build, which aspects to simplify and which to leave in their full complexity—all decisions that parallel a Reggio teacher’s material selections.

## Invitations to Investigate

Reggio environments don’t tell children what to learn; they invite investigation through carefully arranged provocations. A collection of natural materials on a light table, mirrors arranged to create reflections, translucent papers of varying colors—these materials suggest possibilities without prescribing outcomes.




Microworlds function identically. As I describe them, they serve as “an invitation to investigate specific research questions or as a platform for launching new learning adventures” (Stager, 2025). The POLYSPI microworld invites investigation of the relationship between angle and form. Word-action microworlds invite exploration of language as a tool for creation and control. Physical computing with robots invites inquiry into the relationship between abstract commands and concrete actions.




The key, in both Reggio environments and microworlds, is that the materials are provocative without being prescriptive. They embody what educators call “loose parts”—elements that can be combined, rearranged, and repurposed according to children’s emerging interests and questions. When Watt emphasizes that microworlds should be “extensible in the normal Logo way—students add their own procedures to the microworld as they go along” (Watt, 1985), he’s describing exactly this quality of open-ended materials that support children’s own investigations.

## Documentation and Iteration

Reggio practice emphasizes careful documentation of children’s work—not for assessment but to make thinking visible, to support reflection, and to inform the teacher’s ongoing preparation of the environment. Teachers observe how children interact with materials, what captures their attention, what frustrates them, and use these observations to refine their offerings.

This practice maps directly onto the iterative design process I advocate for microworld creation. Teachers observe students working in the microworld, notice where they struggle or soar, and refine accordingly. When Elaine Willis modified the SHOOT game to focus on tossing litter into a trash barrel after observing her students’ values and responses, she was practicing exactly the kind of responsive environmental design that characterizes Reggio practice (Watt, 1985).

The parallel extends to how both approaches value children's productions. In Reggio schools, children's work is displayed not as decoration but as documentation of thinking, as evidence of learning processes, as provocations for further investigation. Similarly, the designs students create in microworlds—the spirals from POLYSPI, the houses from the House microworld, the behaviors programmed into robots—become objects of reflection and starting points for new questions.

## The Teacher as Preparer of Environments

Perhaps most importantly, both approaches position the teacher not primarily as instructor but as designer of rich learning environments. The Reggio teacher's morning work involves arranging materials, considering what might provoke specific children's interests, preparing spaces that invite particular kinds of investigation. This is precisely what a teacher does when creating a microworld.

As I emphasize, "the ability to plant the smallest seed required to grow the most beautiful garden is at the heart of teaching with microworlds" (Stager, 2025). This metaphor captures the Reggio spirit perfectly—the teacher doesn't construct children's learning but rather prepares conditions in which it can flourish. The microworld, like the Reggio atelier, represents the teacher's pedagogical vision made material, embodied in an environment that children can inhabit and transform through their own investigations.

This framing helps teachers who come from progressive education traditions see computational thinking not as a foreign imposition but as a natural extension of practices they already value. The atelier moves inside the computer, but the fundamental pedagogy—respect for children's capability, trust in their curiosity, faith in their ability to construct understanding—remains unchanged.

## Breaking Down False Dichotomies

The Reggio-microworld connection also helps dissolve artificial boundaries between domains. Reggio practice doesn't separate art from science, literacy from mathematics, individual expression from collaborative investigation. Similarly, microworlds can span traditional subject boundaries. A word-action microworld is simultaneously about literacy and computational thinking. A robot programmed to seek light engages physics, mathematics, and engineering. The POLYSPI microworld lives at the intersection of art, mathematics, and computer science.

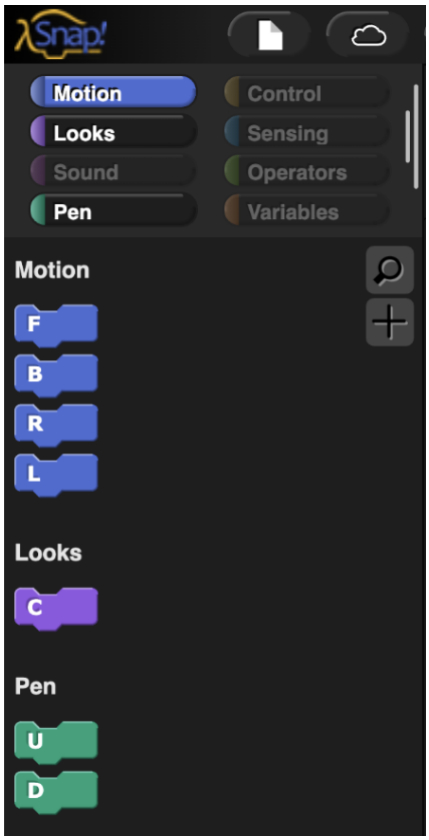
This integration reflects what I describe as bringing "powerful ideas from Bruner, Perkins, and the Reggio Emilia approach to life in your teaching" (Stager, 2025). It's not about using computers to teach traditional subjects more efficiently, but about creating new kinds of learning environments where powerful ideas from multiple domains can be encountered, explored, and connected in ways that mirror how knowledge actually exists in the world—integrated, not compartmentalized.

The Reggio Emilia lens thus provides teachers with a familiar and trusted framework for understanding what might otherwise seem like an alien practice. Creating microworlds isn't about becoming a computer scientist—it's about extending the same thoughtful, child-centered environmental design that characterizes excellent early childhood practice into computational

media. The third teacher can speak in code as well as clay, in sprites as well as paint, in procedures as well as provocations.

## Types and Examples of Microworlds

In my work with teachers, I've seen microworlds take many forms, each designed to make specific powerful ideas accessible to learners at different developmental stages and with varying abilities.



*SNAP! Simple Turtle Graphics Microworld*

**Word-Action Microworlds** serve young children just learning to read. As Watt describes, "typing a word on the keyboard produces an action on the TV screen" (Watt, 1985). A farm microworld might display a pig when the child types "PIG," then make it fly when they type "FLY." These environments parallel how toddlers learn spoken language—through commands that produce observable results. Crucially, as Watt notes, "the objects and vocabulary of a word-action world must be tailored to the individual user." Bob Lawler created a BEACH microworld for his children who lived near the seashore, while Gerri Sinclair built a city vehicles microworld for her two-year-old son. This customization to the child's actual experience exemplifies how teacher-created microworlds differ fundamentally from one-size-fits-all commercial software.

**Simplified Logo Environments** ease beginners into programming. Molly and Daniel Watt created a microworld for 3-to-8-year-olds that simplified Logo commands: "F 10" replaces "FORWARD 100," making effects more observable. As Watt explains, "the REPEAT command slows the turtle down, and makes it appear to move and turn," making rotations more concrete than conventional Logo (Watt, 1985).

**Mathematical Microworlds** help students grasp concepts that typically come much later in traditional curricula. The POLYSPI microworld, for instance, introduces differential calculus ideas to elementary students through spiral designs. By varying parameters like distance, angle, and change, students discover "the stepping of variables as a powerful idea," which Papert identifies as "an essential component of formal operational thought" (Lawler, 1982). Remarkably, these powerful ideas become accessible to children who have never seen an algebraic equation.

**Science Microworlds** make abstract principles concrete. Dynaturtle microworlds simulate Newtonian physics, allowing students to explore motion without friction. Students program goal-seeking behavior using feedback, discovering how objects respond to forces through direct experimentation rather than passive instruction.

## Benefits for Learning

The power of microworlds lies in how they transform the learning process across multiple dimensions.

**Intrinsic Motivation:** Rather than relying on external rewards or punishments, microworlds engage students through inherently interesting phenomena. Papert argues that microworlds solve "the central problem of education" by creating environments where the challenge is to formulate "so clear a presentation of their elements that even a child can grasp their essence" (Papert, 1987). When a six-year-old discovers that speed zero means standing still is a form of motion, the excitement is self-sustaining.

**Personal Appropriation:** Microworlds allow students to make knowledge their own. Rather than absorbing pre-packaged information, learners actively construct understanding through experimentation. This aligns with Papert's mathetic principle of making new ideas your own through playful making. To Papert, mathetic principles were about understanding how people learn most effectively by determining what makes knowledge learnable and teachable.

**Debugging as Learning:** When programs don't work as expected, students must diagnose and fix problems. This debugging process develops metacognitive skills—thinking about thinking—as students learn to trace the logic of their own constructions.

**Transfer of Powerful Ideas:** Well-designed microworlds help students grasp concepts that transfer broadly. The variable manipulation learned through POLYSPI connects to algebra, calculus, and programming. The feedback loops explored in science microworlds illuminate principles in biology, economics, and engineering.

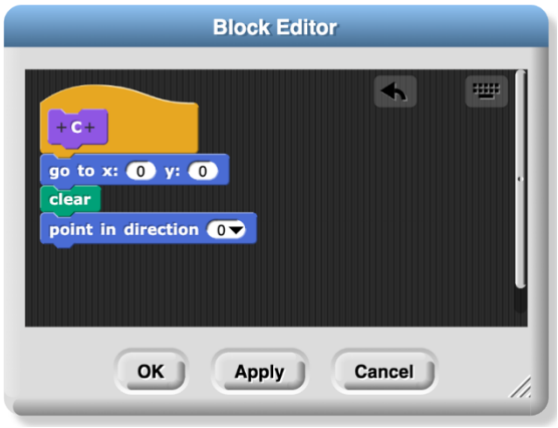
> "A microworld is a small, safe, well-defined learning environment that includes intriguing phenomena, powerful ideas, and meaningful connections to the learner's broader world. Learning in a microworld occurs through open-ended exploration rather than through focused instruction. Logo itself is a mathematical microworld for learning geometric concepts. By creating shapes and patterns, and by repeating and combining them, a learner can discover and apply important geometric principles. In this context, the teacher becomes a helper and guide rather than the sole source of information and instruction. A microworld permits the user to learn by being a mathematician, a poet, or an artist, rather than by merely learning about those subjects. Seymour Papert described learning in a microworld as being more like getting to know a person than like learning a set of facts about that person." (Watt & Watt, 1986).

Simplifying learning experiences through the provision of a microworld is a manifestation of one of Bruner's most durable ideas. When I want four-year-olds to explore a similar domain as a middle schooler or use the same tool as a graduate student, I may create a microworld.

"We begin with the hypothesis that any subject can be taught effectively in some intellectually honest form to any child at any stage of development." (Bruner 1960)

## Essential Elements of Microworlds

Understanding what makes a microworld effective helps teachers both evaluate existing tools and create their own. Several essential elements distinguish genuine microworlds from the broader category of educational software.




*Building a simple CLEAR block to wipe the screen & return the turtle to home in SNAP!*

**Bounded Simplicity Containing Neat Phenomena:** Effective microworlds are carefully constrained—they focus on a manageable domain—but within those constraints, they contain what Lawler calls "neat phenomena": things that are inherently interesting to observe and interact with (Lawler, 1982). The turtle's movement is simple; what students can discover about geometry through that movement is not.

**Manipulable Objects Embodying Powerful Ideas:** Microworlds must give learners something to act upon. As Lawler notes, microworlds should provide "a crisp model" of important concepts (Lawler, 1982). The POLYSPI procedure doesn't teach variable separation—it is variable separation made visible.

**Consistency Enabling Exploration:** Students must be able to trust that the microworld behaves according to consistent rules. Without this predictability, experimentation is impossible—if results are arbitrary, there is nothing to discover. This consistency is what enables the "probing" that Feurzeig describes as central to microworld learning.

**Extensibility Supporting Ownership:** Microworlds should be "extensible in the normal Logo way—students add their own procedures to the microworld as they go along" (Watt, 1985). This allows learners to grow beyond initial scaffolding, transforming from users into builders. The microworld becomes theirs.

**Teacher Support Without Prescription:** While Feurzeig emphasizes that microworlds themselves don't teach, he's equally clear that teacher involvement is essential: "skilled teachers overcome these deficits by providing the guidance and support that make microworld experiences productive for their students" (Feurzeig). The key is that this support enhances exploration rather than replacing it.

**If You Make Simple Ideas Easy to Do, You Make Complexity Possible:** Whenever you teach anything, the pedagogical art involves determining an acceptable level of opacity. Whether for purposes of expediency or making a concept more concrete, microworlds can help students level up by hiding abstractions. The good news is that in the sorts of software environments we are talking about (descendants of Logo), what's hidden from students can easily be made visible as understanding develops.

Understanding these essential elements helps teachers evaluate existing educational software (Is this really a microworld or just a tutorial in disguise?) and design their own microworlds that embody these powerful principles. When all these elements come together—bounded simplicity containing neat phenomena, manipulable objects embodying powerful ideas, consistency enabling exploration, extensibility supporting ownership—microworlds create uniquely powerful learning environments.

**Invites Teachers to Think about Thinking** Building a microworld requires careful attention to a student's needs, interests, and zone of proximal development. Observing and thinking about a student's thinking strengthens a teacher's thinking while undoubtedly improving their practice.

> "You can't think about thinking without thinking about thinking about something."
> (Papert 2005)

## What Teachers Can Do

Microworlds are not self-contained pedagogical solutions—they require thoughtful teacher involvement to realize their potential. Several practical strategies help teachers use microworlds effectively.

**Customizing Learning Environments:** Teachers can create microworlds tailored to their curriculum or students' needs. A teacher planning a geometry unit might build procedures for drawing regular polygons, allowing students to explore properties without getting bogged down in basic turtle commands or gain experience with variables without being taught. This is what I mean when I describe microworlds as "a form of scaffolding that gives flight to a student's thinking" (Stager, 2025).

**Cross-Curricular Integration:** Microworlds naturally bridge subject boundaries. A teacher might develop tools that allow first graders to combine newly acquired reading skills with the list-processing features of Logo. The goal isn't teaching Logo—it's using Logo to enrich literacy learning.

**Differentiated Instruction:** By varying the complexity of available tools, teachers can support diverse learners. When using simple turtle graphics commands, a House microworld provides pre-made procedures for doors, windows, and trees, allowing students to focus on spatial reasoning and turtle states without struggling with complex drawing commands (Watt, 1985).

**Professional Development:** Creating microworlds develops teachers' own computational thinking. Watt argues that "training teachers to make microworlds can make a difference" by helping them see Logo "as an environment for learning rather than as a subject to be taught" (Watt, 1985). This shift in perspective proves transformative and is essential to collaborating with AI in the development of personally meaningful software solutions.

## Design Principles

Creating effective microworlds requires attention to several key principles.

**Embody Powerful Ideas:** Papert offers four criteria: powerful ideas should be simple, general, useful, and syntonic (Papert, 1980). For example, a POLYSPI microworld where students experiment with values for size and length, succeeds because variable separation is fundamental, broadly applicable, and connects to students' experience with patterns.

**Make State Variables Visible:** Lawler emphasizes that "all the state variables should be represented in a visible, obvious way" (Lawler, 1982). If students can't see what's changing, they can't understand the system's behavior.

**Balance Constraint and Freedom:** Effective microworlds limit complexity without restricting creativity. The simplified Logo commands "F" and "R" (shorthand for FORWARD AND RIGHT) constrain options but expand what beginners can accomplish.

**Support Extension:** Microworlds should be "extensible in the normal Logo way—students add their own procedures to the microworld as they go along" (Watt, 1985). This allows learners to grow beyond initial scaffolding. In other words, students can build upon the "building blocks" provided to create their own tools and artifacts.

**Consider the Audience:** As Lawler notes, designers must account for "the interests and knowledge of the particular learner or learners" (Lawler, 1982). A farm microworld works for rural children; city children might need vehicles instead. Microworlds reflect a teacher's understanding of their students.

## Challenges and Future Directions

Despite their promise, microworlds face obstacles. Papert identifies a pattern where schools adopt innovations but transform them "into innocuous forms" (Papert, 2002). Logo itself experienced this transformation: what began as an open-ended exploratory environment was often reduced to drawing exercises with predetermined outcomes.

The challenge for educators is to resist this domestication of microworlds—to maintain their essential character as open, explorable, extensible environments even as they are integrated into structured curricula. This requires ongoing professional development, administrative support, and a willingness to embrace the productive uncertainty that genuine exploration entails.

There are microworlds inside microworlds. Software environments, like Logo dialects, Scratch, SNAP!, or MakeCode are microworlds in which more constrained or enhanced microworlds may be created.

MicroWorlds continue to be relevant. From Scratch and SNAP! to physics simulations and data science notebooks, contemporary tools embody microworld principles while expanding their reach into new domains. The underlying insight—that learners need rich, bounded, manipulable

environments to construct powerful understanding—remains as relevant as ever. However, microworlds are not just for limiting what a learner can do, they may also supercharge what's possible by making complexity accessible and concrete.

## Primitives and Pseudo Primitives

The words or blocks built into a programming system are called primitives. These operations are the fundamental vocabulary for constructing computer programs. In traditional Logo environments, primitives fall into two categories, commands and reporters (sometimes called functions). These primitives may include inputs. FORWARD, RIGHT, REPEAT, CLEAN, IF, SETPOS are all examples of commands. RANDOM, SUM, FIRST, HEADING are reporters that return, or report, a value that is then "caught" by a command as input.

Logo, Lynx, Scratch, SNAP!, Makecode, microBlocks, TurtleStitch, and other procedural languages allow users to define their own procedures and subprocedures made of primitives. These procedures function in the environment just like primitives, but are not built into the system. These procedures save in files containing the projects created using them.

Such user-created procedures may be thought of as pseudo primitives and form the basis for a teacher-created microworld. Traditional versions of Logo allowed for these pseudo-primitives to be added to the programming language itself, to be universally available across projects and files. SNAP! makes that possible, while most other programming environments require a user to load a specific file to get started working with the building blocks of that microworld. In the case of young students, that file can be loaded by the teacher. SNAP! allows teachers one more level of sophistication, the ability to have only teacher-created blocks (primitives) visible to users – furnishing the most perfect form of microworld.

## A Parallel Tradition: Parsons Problems in Computer Science Education

Microworld thinking, it turns out, did not stay confined to the Logo community. A parallel tradition in computer science education research independently arrived at strikingly similar conclusions through a different route. Researchers studying how novices learn to program developed a family of exercises called Parsons Problems—code puzzles in which learners arrange scrambled lines of working code into a correct sequence—and found that removing the burden of syntax recall freed students to focus on precisely the kind of logical reasoning that microworlds have always been designed to cultivate. The convergence is not coincidental: both traditions are responding to the same fundamental truth about how people learn complex things. Teachers interested in exploring this approach will find that SNAP!, the block-based programming environment already popular in K–12 classrooms, allows them to create Parsons-style puzzles using actual code—making the idea immediately accessible without requiring any new tools or technical expertise. Teachers who want to go deeper will find a fuller treatment of Parsons Problems, including their history, theoretical grounding, and classroom practice, in Appendix A.

## Conclusion

Microworlds represent one of education's most powerful and underutilized ideas. By creating bounded environments where students can explore, manipulate, and discover—rather than simply receive—microworlds transform the learning process from passive absorption to active construction.

For teachers, the microworld framework offers both a practical toolkit and a guiding philosophy. It suggests that our role is not primarily to deliver information but to design environments where powerful ideas become discoverable. It reminds us that the most effective scaffolding is the kind that eventually becomes unnecessary—that gives students the tools they need to outgrow it.

In a landscape crowded with educational technology that promises to teach, microworlds offer something rarer and more valuable: environments that invite students to learn.

## References


Bruner, J. S. (1960). *The process of education*. Harvard University Press.

Cohen, R., & Geva, E. (1986). Fostering pre-planning and debugging skills in young children through increasingly complex Logo microworlds. Logo 86 Proceedings.

Feurzeig, W. (n.d.). Towards intelligent microworlds. BBN Laboratories.

Lawler, R. W. (1982a). In the lap of the machine. Boston Review.

Lawler, R. W. (1982b). Designing computer-based microworlds. Byte.

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. Basic Books.

Papert, S. (1987). Microworlds: Transforming education. In C. Kiesler, S. Siegel, & J. McGuire (Eds.), Artificial intelligence and education: Learning environments and tutoring systems (pp. 79–94). Lawrence Erlbaum Associates.

Papert, S. (2002). The turtle's long slow trip: Macro-educological perspectives on microworlds. Journal of Educational Computing Research, 27(1&2), 7–27. https://doi.org/10.2190/2DYL-C83X-8FVD-1L8L

Papert, S. (2005). You can't think about thinking without thinking about thinking about something. *Contemporary Issues in Technology and Teacher Education, 5*(3/4), 366-367.

Perkins, D. (2009). Making learning whole: How seven principles of teaching can transform education. Jossey-Bass.

Stager, G. S. (2025). Microworlds and programming, microworlds as vehicles for learning: Introducing microworlds as means of providing scaffolding for learners with specific needs and for teachers to increase their computational fluency. Constructionism Conference Proceedings, 8/2025, 593–594. https://doi.org/10.21240/constr/2025/99.X

Watt, D. (1985). Teacher-made microworlds or training teachers to use Logo vs. training them to teach it. In Proceedings of the Joint Conference of Educational Computing Organization of Ontario and the Association for Educational Data Systems (ECOO/AEDS), Toronto, Ontario.

Watt, M., & Watt, D. (1986). Teaching with Logo: Building Blocks for Learning. Addison-Wesley Publishing Company.

## Appendix A: Parsons Problems: A Parallel Tradition

The main body of this paper makes the case for microworlds on the basis of a rich constructionist tradition stretching from Papert's MIT laboratory through decades of classroom practice. But constructionism is not the only body of work that has arrived at microworld-like conclusions. Computer science education researchers, working largely independently and motivated by the practical challenges of teaching programming at the high school and university level, developed a family of exercises that embodies many of the same principles—not because they were drawing on Papert and Lawler, but because they were responding to the same fundamental realities about how learners engage with complex material.

That family of exercises is known as Parsons Problems, and while they are most prevalent in AP Computer Science courses and higher education, the thinking behind them is directly applicable to K–12 classrooms—particularly for teachers who use SNAP!, which allows Parsons-style puzzles to be built from actual code blocks. Understanding Parsons Problems is worthwhile not only as practical technique but as independent confirmation that the microworld tradition was onto something real. When two separate research communities, working from different theoretical starting points, arrive at compatible answers, the convergence itself is evidence worth taking seriously.

### What Are Parsons Problems?

A Parsons Problem presents learners with all the correct lines of code needed to solve a problem, but scrambled out of order. The learner's task is to arrange those lines into a correct, working sequence. In more challenging versions, "distractor" lines are included—plausible-looking code that does not belong in the solution—requiring learners to evaluate, not just sequence.

Think of it as the difference between solving a jigsaw puzzle and painting from a blank canvas. Both are valuable experiences, but they develop different skills and suit different moments in a learner's journey. Parsons Problems create a bounded, manipulable environment in which the complexity of syntax is removed so that the learner can focus entirely on logic, structure, and sequencing—precisely the kind of "neat phenomena" that Lawler (1982) identified as essential to effective microworld design.

### A Brief History

Parsons Problems were introduced by Dale Parsons and Patricia Haden in a 2006 paper presented at the Australasian Computing Education Conference. Their central insight was simple but consequential: if the cognitive load of remembering syntax prevents beginners from reasoning about program logic, then temporarily removing the syntax burden should free learners to think more clearly about structure. This insight drew on a well-established body of research in cognitive science. John Sweller's foundational work on cognitive load theory demonstrated that working memory is limited, and that instructional designs which reduce unnecessary mental effort—what Sweller called extraneous cognitive load—leave more capacity available for the kind of deep processing that produces genuine learning (Sweller, 1988). Parsons Problems apply this principle

directly: by providing the code and asking only for arrangement, they strip away the extraneous load of syntax recall and focus all available mental resources on computational reasoning.

Early studies confirmed this intuition, finding that students who practiced with Parsons Problems developed coding ability faster than peers who wrote code from scratch—a finding reminiscent of Papert's observation that the right constraints, far from limiting learning, can accelerate it.

In the years that followed, researchers expanded and refined the approach. Barbara Ericson and her colleagues at Georgia Tech conducted influential studies on adaptive Parsons Problems—versions that adjust their difficulty in response to student performance—and integrated them into Runestone Academy, a widely-used open-source platform for computing education. By the 2010s, Parsons Problems had become a standard tool in computing education research, appearing in university courses, MOOCs, and K–12 curricula alike.

## The Theoretical Connection to Microworlds

Parsons Problems are not typically described in the microworld literature, but they share its most important theoretical commitments. Like a well-designed microworld, a Parsons Problem is:

**Bounded but rich.** The problem space is constrained—learners work only with the lines provided—but within that space, genuine thinking is required. There is no single mechanical path to the solution; learners must reason about dependencies, loops, conditionals, and flow.

**Manipulable.** Learners interact directly with the material, moving pieces, testing arrangements, and revising their thinking. This hands-on engagement aligns with Papert's constructionist principle that learning is most powerful when learners are making something—even when what they are making is an arrangement rather than an original composition.

**Syntonic.** The problems can be designed around contexts that are personally meaningful to students—drawing shapes, writing simple games, solving everyday problems—connecting new computational ideas to what learners already know and care about.

**Non-prescriptive Exploration.** Like Feurzeig's description of Logo microworlds, a Parsons Problem does not explain itself or walk learners through predetermined steps. It simply presents a situation and invites reasoning. As Feurzeig observed of microworlds generally, they "can be probed, their behavior explored, their structure understood, their underlying operation discovered."

Where Parsons Problems extend the microworld idea is in their particular attention to cognitive load. Sweller and his colleagues later refined the theory to distinguish between intrinsic load—the inherent complexity of the material being learned—and extraneous load—complexity introduced by poor instructional design (Sweller, van Merriënboer, & Paas, 1998). Parsons Problems are essentially an instructional design choice that reduces extraneous load without reducing intrinsic load: the programming logic remains genuinely challenging, but the burden of syntax is lifted. By temporarily setting aside the burden of recalling and typing syntax, they create the kind of focused, manageable problem space that Lawler (1982) described as essential: complex enough to be interesting, simple enough to be productive.

## Parsons Problems in the Classroom

For K–12 teachers, Parsons Problems offer several practical advantages that complement a microworld-centered approach to computing education.

### *A Day in Ms. Rivera's Class*

It is a Tuesday morning in a fifth-grade classroom, and Ms. Rivera is introducing her students to loops for the first time. Yesterday, they spent the class period in Logo, commanding the turtle to draw squares: first by typing FORWARD and RIGHT four times each, then discovering, with considerable delight, that they could wrap those commands in a REPEAT block and watch the turtle do the same work in a fraction of the code. The exploration was noisy and productive, full of wrong turns and happy accidents.

Today, Ms. Rivera wants to see whether that intuition has taken root. She hands each student a small envelope. Inside are seven strips of paper, each containing one line of a short Logo program that draws a row of triangles. The strips are shuffled. One of them is a distractor, a line that looks plausible but does not belong.

The room gets quiet in a different way than it does during a test. Students spread their strips on their desks and begin to reason aloud, with themselves, and with each other. "The REPEAT has to come before the stuff inside it." "Wait, what does this line even do?" "I think this one is the fake." Ms. Rivera moves through the room, not answering questions directly but asking her own: "What do you think will happen if that line goes there?" "How would you check?"

When students settle on an arrangement, they copy it into Logo and run it. The turtle either draws the row of triangles or it does not, and in either case, something useful happens. A correct arrangement produces satisfaction and a moment worth pausing to celebrate: you read that program and understood it well enough to reassemble it. An incorrect arrangement produces a result worth investigating and debugging: what did the turtle actually do, and why?

What Ms. Rivera notices, circulating through the room, is not primarily who got it right. She is watching how students reason: who is confused about the relationship between the repeat count and the shape, who spotted the distractor immediately, and who needed to try it in the program first to be sure. By the time the class reconvenes, she knows exactly which ideas to build upon and which students need a different entry point. No quiz could have told her as much.

**They lower the barrier to entry without lowering the intellectual standard.** Students who might be intimidated by an open coding task can engage meaningfully with a Parsons Problem from their first day of instruction. This is not about making things easy. It is about making the hard things accessible.

**They make student thinking visible.** When a student arranges lines of code incorrectly, the teacher gains insight into their mental model of the program's flow. This is far more diagnostically useful than a blank page or a syntax error. The arrangement itself becomes a window into the learner's understanding. That said, observing children while programming, engaging in conversation, or reading their code may be even more informative, less intrusive, and more authentic.

**They scale gracefully.** A teacher can design Parsons Problems that range from very simple (three or four lines, no distractors) to genuinely challenging (ten or more lines, multiple distractors, nested structures). The same basic format serves beginners and more advanced learners alike.

**They are easy to create.** Unlike many educational tools that require significant technical infrastructure, a Parsons Problem can be constructed with nothing more than a working program and a willingness to cut it into pieces. SNAP! makes the creation of Parsons Problems simple. Free tools such as js-parsons and Runestone Academy make it straightforward to create versions in other languages.

**They pair naturally with other microworld activities.** A teacher might introduce a new programming concept through open-ended turtle graphics exploration, then consolidate that learning with a Parsons Problem that requires students to sequence a related program. The two activities can be complementary: the microworld builds intuition, the Parsons Problem builds structure.

## The Assessment Temptation

Since Parsons Problems produce a clear, scorable outcome (the arrangement is either correct or it is not), they can be tempting to reach for as assessment instruments, particularly in schools where teachers feel pressure to document and grade student progress in computing. That temptation is worth examining carefully as if assessment is necessary at all.

Parsons Problems may function as formative assessment. When a student places a loop's body outside the loop, or sequences a variable reference before its definition, the arrangement itself reveals something real and useful about their thinking. A teacher who notices these patterns gains far more diagnostic insight than a simple correct/wrong score provides and can respond with targeted questions or a well-chosen follow-up exploration. In this sense, Parsons Problems make student thinking visible in ways that are genuinely useful for instruction.

The danger lies in treating them as evidence of broader mastery or deficit. Parsons Problems assess recognition and arrangement: a student must identify which line belongs where among options already provided. Writing code from scratch requires something different and harder: retrieving syntax from memory, generating logic without a scaffold, and sustaining that thinking across an entire program. A student can perform well on a Parsons Problem and still struggle significantly when faced with a blank screen. These are related but distinct capabilities and conflating them does students a disservice. A freer more flexible programming environment may also serve student

bricoleurs who learn to program by doing so in a generative fashion, engaged in conversation with their code.

This is not a minor technical distinction. In a constructionist classroom, the goal is for students to experience themselves as capable makers and thinkers, people who can bring something new into the world through code. Parsons Problems can build toward that experience, but they cannot confirm it. Using them as the primary measure of programming ability is a little like assessing a student's writing ability by asking them to unscramble sentences. The task is related to writing, it reveals something about language understanding, but it does not tell you whether the student can compose.

The most defensible approach is to treat Parsons Problems as one source of formative evidence among many: useful for surfacing misconceptions and building structural understanding, but always in service of the larger goal of students who can sit down with an interesting problem and make something.

### A Note on What Parsons Problems Are Not

It is worth being explicit about the limits of the approach, in the spirit of Feurzeig's reminder that microworlds "do not teach." Parsons Problems, like microworlds, are not a delivery mechanism for instruction. They do not replace the open-ended exploration, student-driven projects, and rich classroom conversation that lie at the heart of constructionist teaching. Used well, they are one tool among many: a particularly well-designed on-ramp that helps learners build the confidence and structural understanding they need to engage more freely with open creative work. Used poorly, they can become just another form of drill, disconnected from genuine making and thinking.

The goal, as always, is to give students the experience of being competent, reasoning programmers, and then to get out of their way.

### Additional References


Ericson, B. J., Margulieux, L. E., & Rick, J. (2017). Solving Parsons problems versus fixing and writing code. Proceedings of the 17th Koli Calling International Conference on Computing Education Research, 20–29. https://doi.org/10.1145/3141880.3141895

Parsons, D., & Haden, P. (2006). Parson's programming puzzles: A fun and effective learning tool to introduce software engineering. Proceedings of the 8th Australasian Computing Education Conference (ACE 2006), 157–163.

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. Cognitive Science, 12(2), 257–285. https://doi.org/10.1207/s15516709cog1202_4

Sweller, J., van Merriënboer, J. J. G., & Paas, F. G. W. C. (1998). Cognitive architecture and instructional design. Educational Psychology Review, 10(3), 251–296. https://doi.org/10.1023/A:1022193728205


## Appendix B: Microworld Creation Tutorials

The following documents provide clear step-by-step instructions for creating microworlds in Turtle Art and SNAP! There is an additional handout on how to create a microworld for the Finch robot, an excellent modern floor turtle for classroom use.

# Creating a Microworld in Snap!



Snap! is a visual, block-based programming language designed for educational purposes, especially to introduce learners of all ages to computer science and computational thinking in a powerful and accessible way. It is an extension of the Logo family tree and may be thought of as Scratch for computer scientists.

You can design a microworld in Snap! due to its flexibility and large number of primitives. Snap! allows you to hide primitive blocks built into the system and add your own blocks to the programming environment. Some microwords may be intended to assist learners by eliminating confusion or distraction caused by too may blocks to choose from.

Creating a microworld with a limited number of blocks has obvious applications for working with young children or learners with special needs. In such cases, the Snap! environment is customized in a prosthetic fashion.

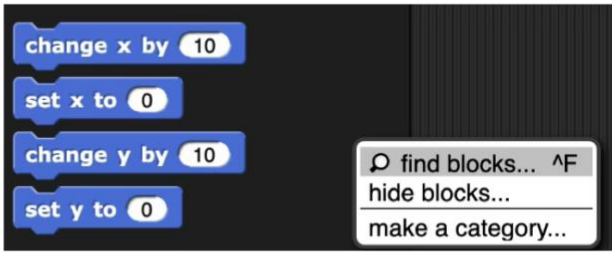
A microworld may also add new high-level blocks created to perform complex tasks or simplify functionality. One might imaging a trigonometry microworld or fractal microworld in which new blocks (super procedures) are added to Snap! to allow users to engage with powerful ideas without the need to build the blocks themselves or understand the underlying code.

Best of all, customizing the Snap! environment represents a form of toolmaking. Building one's own tools is a human superpower that allows us to be more efficient or solve more complex problems.

**How to hide blocks**

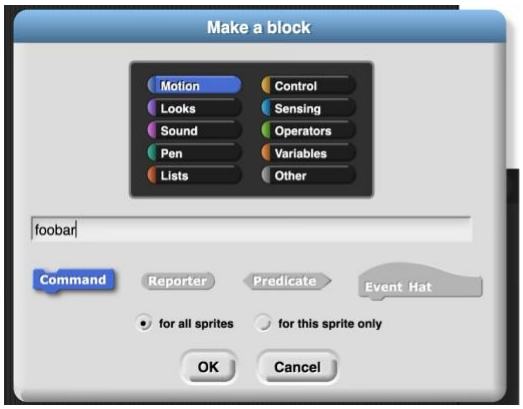
1. Open Snap! at snap.berkeley.edu
    1. login to your account if you have one
2. Click the File Icon and select `Hide blocks…`
3. Click on the blocks you wish to hide from the system. Those blocks will disappear from your blocks palette.






Alternatively, you can CTRL-Click (Mac) or Right-Click (PC) in the blocks palette and choose `hide blocks`. This allows you to hide blocks from that palette of blocks only, not all palettes.

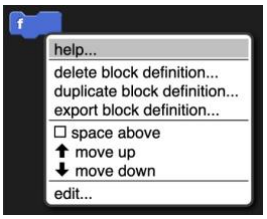
Incidentally, if a block uses another block in its own code, it will still work despite being hidden from view.





**How to add blocks**


1. Click `Make a block` button in the palette
2. Click on the color/category of block you wish to create
3. Determine the function of the block – command, reporter, predicate, event hat
   Name the new block
4. Click the plus sign to add inputs if your block requires them
5. Drag blocks into the block editor to create the definition for that block
6. Click OK
7. Test your block
8. Debug, edit, or improve your block by CTRL-Click (Mac) or Right Click (PC) on the block and select `edit…`




**Share your projects with students or colleagues as a link** (cloud sharing)

1. Make sure you're logged into your Snap! account and your project is saved.
2. Go to File → Open…, select your project in the list.
3. Click the Share button and click "Yes" if prompted
4. The address bar will update to a URL containing your username and project name.
5. Copy that URL and send it to anyone — when they open it, they'll see the most recent version saved to the cloud

Note: Whenever you save a project, even after sharing, the link updates to reflect your new changes.

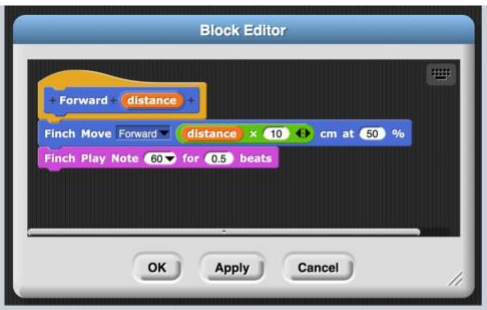
**MakeCode**

*The Invent to Learn Guide to the micro:bit* from Constructing Modern Knowledge Press, features a chapter describing how new blocks may be created in MakeCode for use with the BBC micro:bit, video game programming, and other microcontrollers.

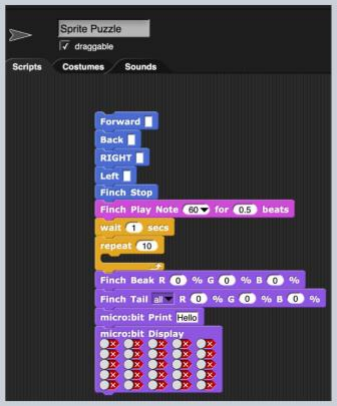
# Creating a Finch Microworld



1. Get a charged Finch robot
2. Start a new project at snap.birdbraintechnologies.com
3. Establish a connection to your Finch
4. Build your own blocks for making the Finch behave like a Logo turtle, blocks for `forward, back, right, left...`
5. Debug
6. Connect all of the blocks you programmed and any other essential blocks and choose Generate Puzzle from the Snap! file menu. This creates a "new" version of Snap! featuring just the blocks you chose. You may also hide irrelevant blocks. (see Snap! microworld handout)
7. Save the project for the use of students.
8. Observe students using your new software.
9. Make adjustments as necessary.




Build your own block with an input




Connect all the blocks you wish to include in your puzzle

Or hide all of the undesired blocks in each palette

Select Generate puzzle from the file menu to create a puzzle

**Challenges**

- Make blocks for moving the turtle and turning by reasonable physical distances
- Make blocks for "driving" the turtle with F, B, R, L and no input
- Create turtle movement blocks where 1 = 10 Finch/turtle steps
- Is it possible to use Finch sensors to make sure your turtle does not run off a table?
- Can you create a block for resetting/clearing a turtle (such as `clean, clear, cg`, or `cs`)?
- Change your blocks to
- Design a version of Turtle Art in Snap!

**Resources**

Check out the turtle graphics/turtle geometry project ideas found in eBooks available at dailypapert.com/logo, including *LogoWorks, Teaching with Logo, Learning with Logo, Turtle Confusion* and *Turtle Speaks Mathematics.* Can you use these ideas in your new Finch Turtle microworld?

# Creating a Microworld in Turtle Art



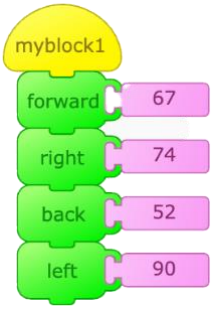
Sadly, Turtle Art does not allow users to hide blocks or customize the block palette. There are ways, however, to create your own blocks and to direct learners to use those specific blocks for the purpose of creating a microworld. playfulinvention.com/webturtleart

**Create a new block**

New blocks are created by putting a "hat" on top of a stack of blocks describing a procedure composed of a list of commands (other blocks).

**Name a new block**

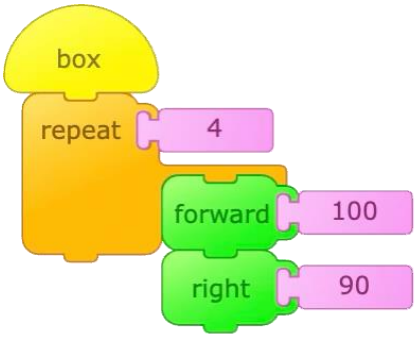
Click on the yellow "hat" and type a one-word name for the new block. Do not use a word already used by a Turtle Art block, or a number, or multiple words. If you wish to name a block *My Block 1,* name it *myblock1.* Most programming languages have limitations and syntactical conventions.

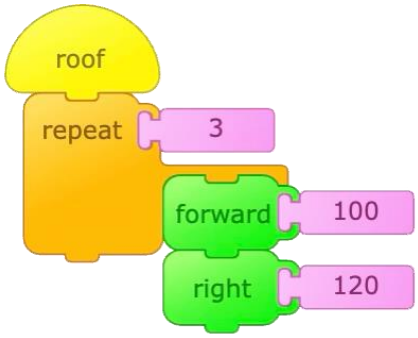


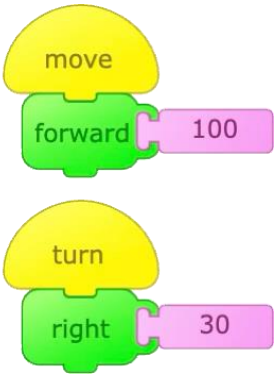

Make sure all the blocks are connected to the naming block (hat) by clicking and dragging the "hat." Every block below the "hat" should remain connected.

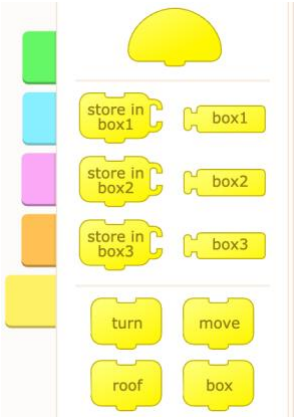
**A simple microworld**

Give learners these four blocks and ask what they can make with them. (there is no right answer). You can print the blocks on card stock or just ask that their creations are limited to the blocks below the line in the yellow palette.



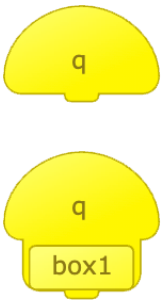



## Adding inputs to a custom block

Turtle Art includes three blocks representing global variables, *box1, box2,* & *box3.* There's a trick for creating blocks with inputs. For example, the `forward` block has an input, so does `right`. To create a block with an input, do the following.

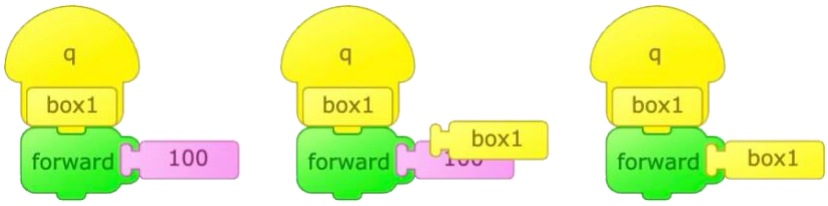
1. Drag a hat/naming block onto the screen or add a hat/naming block to a stack of blocks.
2. Give the stack a name by clicking on the hat block, typing a name, and clicking anywhere else on the screen. You should now see the new block in the (yellow) palette of blocks.
3. Next, drag a `box1`, `box2`, or `box3` block on top of the hat block. It should put a rectangular sign with the name of the input you just added inside the mushroom-shaped hat.
4. That's it. That block now has an input!



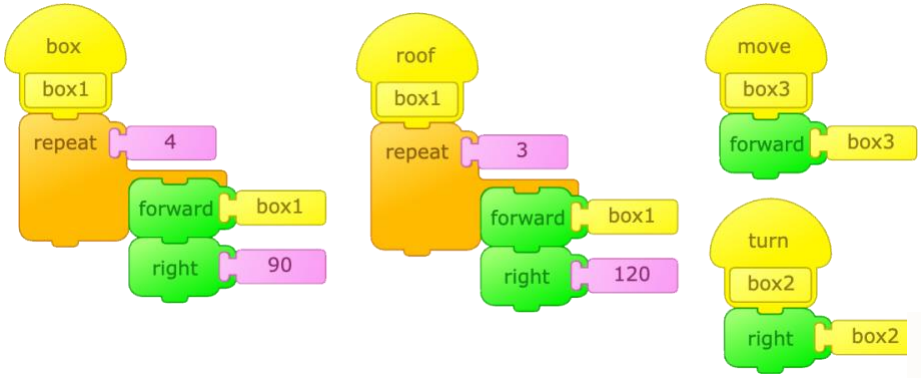

## Using a new block containing an input

The `box1`, `box2`, or `box3` you used as an input may be used to represent a value substituting for a fixed or constant number. For example, instead of `forward 10`, you can say `forward box1`, and the thing inside `box1` will be the value handed to `forward`.

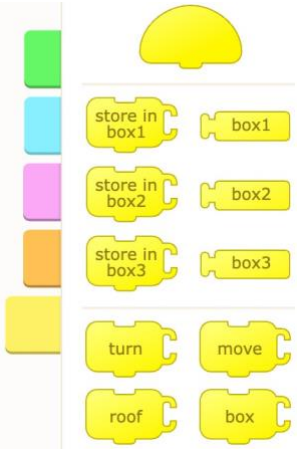
- Dragging an input block over a pink numerical block will replace the value with an input.




## A new microworld




- The input you choose, `box1`, `box2`, or `box3` does not matter since its value is local to the particular block you are creating.




## Deploying your microworld

1. Save the project
2. Open Turtle Art on each computer
3. Drag the project you saved into Turtle Art
4. Mess about with the microworld!