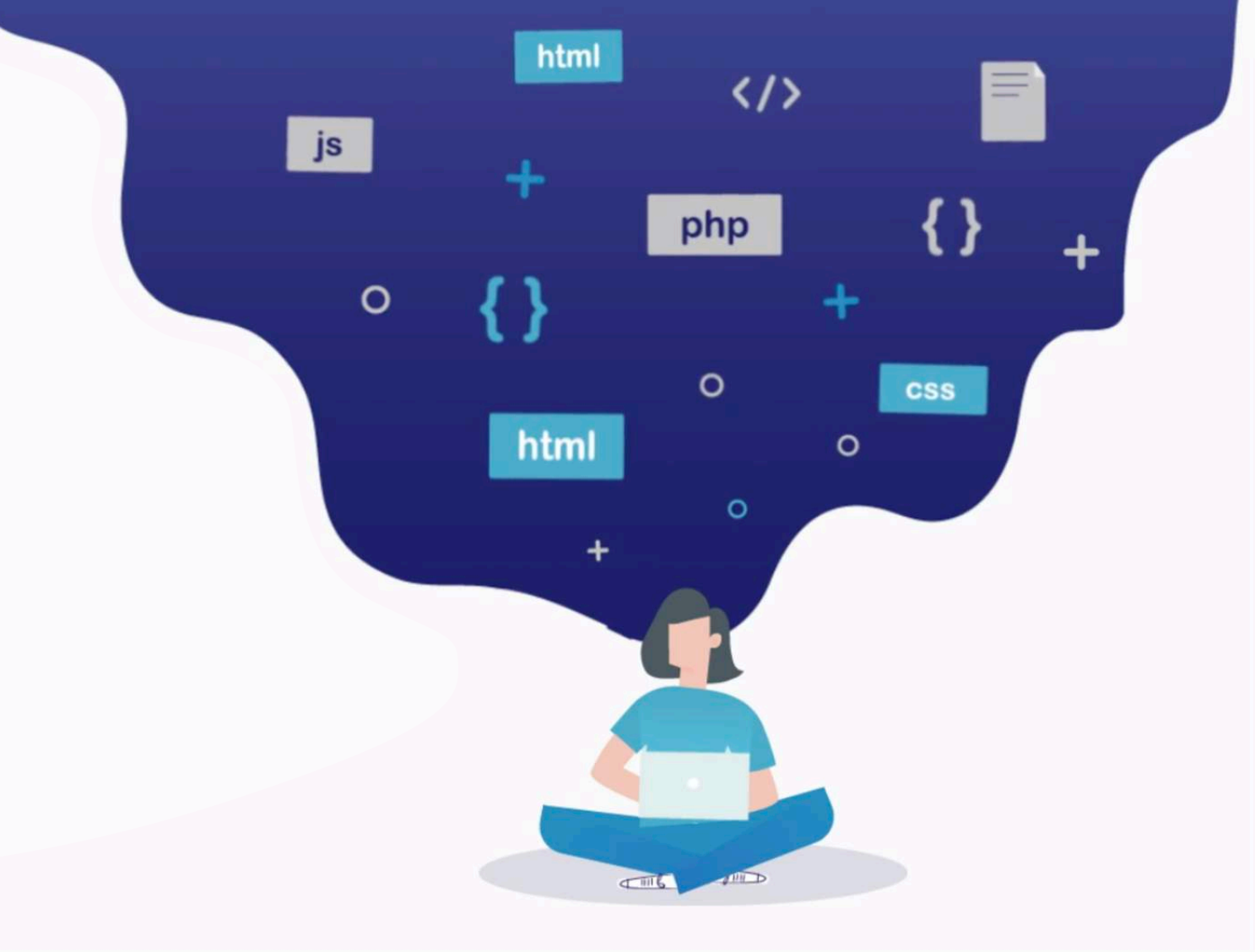


How Best to Prepare
Your Child for a

DIGITAL FUTURE

UCode

www.ucode.com

INTRODUCTION

The world is rapidly transforming. In less than 50 years, computers have gone from room-sized machines with limited applications to pocket computers we carry with us and use on a daily basis.

But our kids don't know this. Today's kids are digital natives. They're surrounded by technology from the start and grow up comfortable using computers, tablets and cell phones from a very young age. This is indicative of the ubiquity of digital technology and the intelligence of our children. Today there's scarcely a moment where they aren't interfacing with some piece of networked, computer-controlled equipment.

Consider an average day. You son or daughter is awakened by an alarm clock app on their phone. They brush their teeth with your sonic, electric toothbrush. Heading downstairs they grab a quick breakfast while texting their friends or catching up on the news off Instagram or a social site ...

Now they are out the door. At school, they engage with computers, tablets, digital whiteboards, networked printers, and a host of other technologies they don't even know is technology. And this is all just what we can see. There's another level of invisible technology that runs the background machinery of our world. It keeps the power grid running smoothly, enables worldwide telecommunications, guides global food production, and on and on.

All of this technology is controlled by sophisticated hardware and software, and all of it had to be designed by engineers, built in high-tech facilities, and coded and maintained by talented groups of programmers. These are the jobs of the future - the jobs that our children will pursue. Those individuals with the necessary skills will be rewarded handsomely and those without face an increasingly uncertain future.

THIS GUIDE IS ORGANIZED INTO FOUR SECTIONS:

1. What Is Coding & Why Kids Should Learn It.

We start with the basics - what is coding and more importantly the Why? If you are familiar with the subject, you can skip this section.

2. Computational Thinking.

As educators we understand that not all kids will grow up to be programmers. But all the good jobs of the future - even creative arts - will require strong computational foundations.

3. How to Start.

What age can a child start from? What is the best language to start with? What to look for in a coding education program? How much should it cost? Isn't this being taught in school? In this section we cover the basic questions that parents often ask.

4. The Learning Progression.

What educational outcome can you expect? How does this benefit my child? What level of skill do they need? In this section we cover the questions that parents should be asking, but rarely do.

Every parent wants their child to do well in life. However, we are moving toward a future where digital technology is reshaping the economy. We already see automation replacing lower skilled workers and as technology advances white collar jobs and professional jobs will be impacted. It is imprtant that our children develop strong computational thinking skills so as to allow them the futures that we as parents wish for them.

What Is Coding and Why Should Kids Learn It?

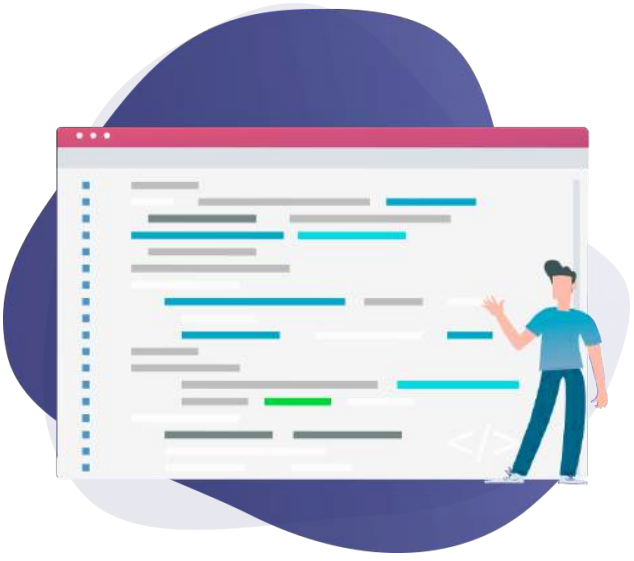
WHAT IS CODING?

Humans use language to communicate. Whether we speak English, Spanish, German, Mandarin, French or any of the other roughly 6,500 languages alive in the world today, the end goal is the same. To share information and experiences with other people.

Computers share information as well. However, computers “speak” a language of 1s and 0s. On and off. The lowest level of language a computer understands is machine code, a purely numerical language. While it’s possible to write programs directly in machine code, the process is extremely challenging and highly prone to errors.

To make life easier, computer scientists have created **higher-level computer languages** that look more like human language. These act as an intermediary between people and machine code.

When these languages are compiled, they are converted into the machine code computers prefer. The process of writing instructions for a computer in one of these assembly languages, and there are quite a lot of them, is called coding.



To get a rough sense of what coding is, imagine it's date night and you're leaving the kids with a new babysitter. You're more than likely going to leave the sitter a detailed list of instructions for how to handle your kids while you're away. It might say things like, "If Bobby asks for juice make sure you put it in the purple sippy cup" and "When it's bedtime, you'll need to get Bobby into his pajamas, brush his teeth, and then read him three books before turning out the light, in that order."

In essence, you're "coding" your babysitter. You're telling him or her what to do, and how to do what needs to be done in the event that certain things happen throughout the night. You're coding a "program" for how to properly take care of your kids. For a piece of technology, a similar instruction might be, "When the user presses button two, open a new window, and display the requested information."

Coding isn't done in English, of course. It's written in a coding language. But the process is the same. We write code to tell computers how to do useful things. Without code, and people that know how to write it, computers would be very expensive paperweights.

WHY SHOULD KIDS LEARN TO CODE?

There are several reasons why kids should learn to code:

1. Coding Teaches Kids to Think

Learning to code involves learning how to think methodically - what is know as "Computational Thinkign". Kids that learn to code learn how to take difficult problems and break them down into their root components, making it easier to find effective solutions. Coding teaches them deep analysis and strong problem-solving skills.

Coding teaches a skill set which is transferable to nearly every aspect of life. Anything that requires procedural, logical thinking can benefit from the computational model of thought that underlies a coding discipline. They become better thinkers, better test takers, better students.

2. Learning to Code Fosters Creativity

When kids can code, they can create the tools they need.

They can write their own programs and design them to their exact specifications. They can take ideas from existing programs and then improve them. Coding frees them from the limited, incomplete software and app choices other people are offering and allows them the power to invent anything they can dream of. Coding makes them creators, not consumers.

3. The Earlier You Begin Teaching Kids to Code, The Better.

Coding is a language, written by people, and understood by computers, and children's brains are primed to learn languages. While new research indicates that this rapid language acquisition ability is retained through a child's teenage years, the general consensus says that true fluency is most easily attained prior to the age of ten[1].

This is why it's so critical that computer science education be made available to children throughout their formative years. There's a fairly narrow window during which their brains are sponges for the syntax and grammar of programming, as well as learning the logical thought processes that are an intrinsic part of coding.

As important as this sort of computational education is, it's only now becoming a standard part of public-school education. In 2013, England became the first country to add coding to its required curriculum, mandating instruction for students ages 5 to 16[2]. Other countries, such as France, Finland, and Singapore followed suit. Japan plans to make computer science a mandatory part of their curriculum by 2020[3].

However, the United States is woefully underperforming much of the rest of the developed world. In suburban areas across the country, less than 45% of public high schools teach computer science[4]. In rural areas, this number is even lower. But tthis is for high school, rarely do we see suhc programs in middle school or elementary school and when we do, they use educational languages like Scratch or Mindstorms, that have limited value ...

4. High-Tech Jobs are the High Paying Jobs of the Future

According to a recent study entitled, "Structural Transformation and the Rise of Information Technology", IT job growth far outstripped growth in less IT-intensive fields between 2004 and 2017 by a large margin[5].

The former grew by 19.5 percent over that period, compared to just 2.4 percent for the latter.

Forbes recently published the results of an inquiry they made into the top ten highest paying industries and, not surprisingly, technology applications took that top two spots[6]. Software & IT Services took number one, with average salaries reaching $104,700 a year. Next in line is Hardware & Networking, with average salaries of $101,100 a year.

As digital technologies insinuate themselves ever deeper into the fabric of society, the need for skilled IT workers will only grow, and as the required skillsets increase in complexity and specialization, salaries will increase at a similar pace. If you want your children to thrive in the information economy, you need to begin preparing them today.

[1] "At What Age Does Our Ability to Learn a New Language Like a Native ...." 4 May. 2018, https://www.scientificamerican.com/article/at-what-age-does-our-ability-to-learn-a-new-language-like-a-native-speaker-disappear/ Accessed 25 Jul. 2019.

[2] "National Curriculum in England: Computing Programmes of Study." *GOV.UK*.

[3] "Coding will be mandatory in Japan's primary schools from 2020 ...." 27 Mar. 2019, https://asia.nikkei.com/Economy/Coding-will-be-mandatory-in-Japan-s-primary-schools-from-2020. Accessed 25 Jul. 2019.

[4] "2018 State of CS Education - Code.org." https://code.org/files/2018_state_of_cs.pdf Accessed 25 Jul. 2019.

[5] "Structural Transformation and the Rise of Information ... - ResearchGate." 28 Oct. 2018, https://www.researchgate.net/publication/325495460_Structural_Transformation_and_the_Rise_of_Information_Technology Accessed 25 Jul. 2019.

[6] " The Best-Paying Jobs And Industries In the US - Forbes." 6 Sep. 2017, https://www.forbes.com/sites/karstenstrauss/2017/09/06/the-best-paying-jobs-and-industries-in-the-u-s/ Accessed 25 Jul. 2019.

II. What is Computational Thinking and Why is it So Important?

As educators we understand that not all kids will grow up to be programmers. But all the good jobs of the future - even creative arts - will require strong computational foundations.

Computational thinking has gone by many names over the years. Terms like algorithmizing, procedural thinking, and engineering thinking have all been used to describe the style of thought used in computational thinking. More recently, the term "design thinking" has come into use.

Computational thinking is a way of mentally modeling complex problems so that they can be deeply understood, and optimal solutions can be arrived at in a efficient manner. Learning to think computationally allows people to optimally solve all sorts of problems, no matter how complicated they are. It helps us to reduce problems to simpler pieces, understand how all of the sub-systems relate to each other, and then engineer the best possible outcomes.

WHAT IS COMPUTATIONAL THINKING?

Computational thinking includes the root word “computation” but it involves far more than just performing mathematical calculations. The term is intended to

capture the style of thought used in designing these calculations. This extends to problem-solving in general. Computational thinking attempts to break down a difficult problem into a series of simpler steps which can be used to arrive at a solution.

This style of thinking found its purest expression in computer science, so it's

worth understanding how the term is used in this context.

Computational thinking starts with abstracting a problem and then reducing it down to an extended series of simple instructions that can be understood by a computer. Computer's aren't smart, and they follow instructions literally. In order to get a computer to carry out a task, you have to tell it, in exacting detail, what it needs to do.

Let's say you're trying to tell a computer how to make a peanut butter and jelly sandwich. You would need to make sure the computer knows that peanut butter goes on one piece of bread while jelly goes on the other. And it needs to understand that a knife spreads both. But it can't start spreading before it puts the PB&J on the bread and it definitely can't forget to open the jars before trying to put the knife in!

There are steps in making a sandwich that we take for granted, but a computer knows nothing, and so you have to break down and explain the process absolutely precisely. This requires very detailed, regimented thinking. It requires you to examine a problem very closely, to make sure no details are missed, and to chart out the logic flow and the tree of branching decisions (grape jelly or peanut butter first?) that are needed to arrive at a final solution.

This is, of course, an oversimplification, but it demonstrates well how computational thinking can help us solve problems and develop rigorous thought processes. To understand computational thinking more precisely, we can look at the four steps involved.

Decomposition

This is the process of breaking down a problem into its constituent steps. We did this with our peanut butter and jelly sandwich. It's noting all the places where decisions are made, what assumptions

exist, and what variables are important to keep track of.

Pattern Recognition

Once we have our problem fully described, we comb through looking for patterns, for repeating actions or other pieces of information that have something in common. This is useful for further simplifying the problem to help point a way toward a solution.

Abstraction

Once we've defined the patterns present, we can begin to abstract the problem, or create a mental model that describes, in as few steps as possible, how to arrive at a solution. Through this process, we're also eliminating unnecessary information. As an example, when you're making a sandwich, you might note the color of the plate, but that isn't important to the process. Any extraneous information that doesn't fit the pattern or isn't needed to solve the problem is discarded.

Algorithms

In this last stage we take everything we know about the problem and write it out as an intricate set of instructions that the computer can follow logically to arrive at an answer. This algorithm takes our mental model and explains it to the computer. It allows the computer to "understand" the problem so that it can quickly churn through possible solutions to find the best answer.

It is this process that makes computational thinking so useful in every academic subject and throughout life. Even if we aren't writing programs for computers, being able to express a problem in the detail needed to write a program, helps us analyze the problem in a very deep way, allowing us to arrive at novel solutions, and to really understand things with unprecedented thoroughness.

WHY IS COMPUTATIONAL THINKING IMPORTANT?

The world is growing in complexity. The issues we're facing, both individually and as societies, are growing in number and in difficulty. Society is increasingly in need of computationally-minded people that can solve complex problems. And the lion's share of these solutions will be technological in nature[7]. Which means increasingly the people capable of thinking computationally.

To ensure that your child fits into this digital world, it's critical that they learn how to think computationally. The best way to do this is to teach them to code. Coding computers is something that even young kids can begin to learn, and the process of writing code is a pure expression of computational thinking.

We stress to parents, that even if your childs "dream" job is as a artist or doctor or lawyer, the domains of creative arts, law and medicine are increasingly digital. Strong compuational thinking foundations are absolutely essential in any future occupation. And they are increasingly pre-requisities for entry into the top universities and private schools.

[7] "The Impact of Digital Technology on Society and Economic Growth - IMF" https://www.imf.org/external/pubs/ft/fandd/2018/06/impact-of-digital-technology-on-economic-growth/muhleisen.htm
Accessed 27 Jul. 2019.

III. How to Start

What age can a child start from? What is the best language to start with? What to look for in a coding education program? How much should it cost? Isn't this being taught in school? In this section we cover the basic questions that parents often ask.

WHAT'S THE BEST WAY TO START?

As we noted previouly, the earlier kids begin to learn to code the better. However, coding does require basic math skills, so it is generally not accessible to students younger than 9 or 10 years of age.

Computationl thinking, on the other hand, is accessible for very young children. For children ages 8 or under, we recommend to start with simple problem-solving exercises or games that allows them to identify and complete simple patterns. We do not recommend apps or purely screen based programs for children under 12. Children of this age lack the abstraction skliis necessary to make these tools effective. While they are great fun for play, they have very limited educational value. We do recommend physical objects - Lego or Bricks. One of our favorites is Little Bits, which is remarkable for kids of this age ...

But for kids 9 years of age and older, the easiest way to begin is with a program that uses a Block interface. There are several

available, but the most popular is Blockly, the successor to Google's *App Inventor*. It is a visual block programming editor that acts as a library for adding drag-and-drop block coding to an app. The name Blockly is based on its user interface which resembles a child's toy box, consisting of multi-shaped, multi-colored blocks.

Blockly works by linking colored blocks of code to a function and rendering the result. Each block type usually illustrates a concept or some type of conditional logic that is used in 'real world' programming languages. Both use a drag-and-drop system with small functional blocks of code, making syntactical errors literally impossible. This has proven to be an effective learning tool for kids who have not yet developed typing skills

Learning to code is generally a difficult task as it commands a high cognitive load for young students, as well as introducing a variety of new concepts that are often difficult for young learners to grasp. Block interfaces simplify this problem - picking a block from a selection is far easier than remembering a word. The block format relies on recognition instead of recall, which in turn facilitates learning at a faster rate.

WHICH PROGRAMMING LANGUAGE SHOULD A BEGINNER START WITH?

There is a debate among established programmers as to the easiest language for novice coders to learn. While most languages share the same fundamental concepts, there are some important differences that should be considered, particularly between Java and Python which are among the most popular.

Python is widely adopted as a first language because of its relatively simple syntax which is based closely on the English language. This means non-programmers can often make sense of a simple script - which is very important for kids.

Java on the other hand, is considered to be far less user-friendly and requires a tighter grasp of core programming concepts before achieving any meaningful output. Java uses 'strict' typing - meaning that the user needs to be precise in their commands and error feedback can often be ambiguous to newcomers. Those who learn with strict programming languages usually have a tougher time learning, but likely end up with a better understanding of core

fundamentals. This is unlike Python which will get the job done without necessarily requiring an-depth understanding which is perfect for new coders.

WHY USE ROBOTS TO LEARN TO CODE?

One, it is fun! Kids have a great time which means that engagement is high and if engagement is high, learning retention is high.

But more importatly, programming concepts can be often be abstract and difficult to understand for young learners. Students brains are not fully developed and they typicaly have difficulty with relating the code on the screen to the objective fo the exercise. They simply do not yet have the necessary abstraction skills.

By giving students control over a physical robot which can easily demonstrate errors students quickly learn the need for clarity and precision in their instructions. It is a bridge that ties what they see on the screen with the actions of the physical object - in this case the robot. Through programming a physical device, students gain an important understanding of how these subjects link together.

There are a number of other benefits inherent to teaching robotics in a classroom-based environment, such as promoting team-building skills, shared problem-solving among students and developing a strong incentive to coordinate toward a common goal.

HOW DO I TRACK MY STUDENTS PROGRESS?

Feedback and examination are critical to the learning process; it is essential to evaluate whether the educational goals and standards of the lessons are being met. Curriculum-Based Measurement (CBM) is a form of academic assessment which is commonly used by teachers to shed light on students progress in academic areas.

Unfortuntely there are very few on-line or after school programs that are curriculum based. Yes, many will claim to have a curriculum, but these are not curriculum in the traditional sense of having a set of concepts that they teach and evaluate against. What they offer are "project" based learning experiences that are loosely combined and offered as a curriculum.

This is particularly true of "apps"and on-lne programs. The only one that we can

recommend is CodeSpark, as it is designed to build computational thinking skills. The rest of the apps in the market are really for "play", even if they claim to "teach kids to code". And play is great, but do not expect any real learning outcome.

We would note that Ucode and Cornell University have a program to measure computational thinking and coding literacy among students. It is the first of its kind in the United States and we belive beyond.

TIME & MONEY

At UCode, students typically require 50-60 hours of time to complete Introduction to Python course and an additional 80 hours to complete Intermediate Python. At this stage of their learning journey, their computational thinking foundations will be well developed.

If they wish to continue into Advanced Python it will require an additional 80 hours. Upon completing the Advanced Python course, they will have completed coursework equivalent to a CS100 course at a leading university.

Typically, we see after school coding programs charging $250 per month for a 60 minutes class each week (roughyl $60 per hour). UCode is signifcantly less expensive (roughyl 50% on an hourly basis) with a class length of 90 minutes.

EDUCATIONAL OUTCOMES

Learning to code teaches kids how to think computationally. It helps them break down difficult problems into more manageable pieces and then apply a range of possible solutions to find the most effective one. It trains a logical way of approaching situations. This ability to think computationally will:

- ***Fosters Creativity.***

  Coding is accomplished through experimentation, trying novel ways of approaching a problem. Code frequently fails, but these failures are valuable teaching moments. Kids discover new ways of thinking about problems when they try a solution and it doesn't work the way they expected. Coding builds resilient, flexible, creative minds.

- ***Improves Academic Performance.***

  Learning to code involves learning

how to think logically. Kids learn how to tenaciously apply themselves to a problem until they find a workable solution. This teaches organizational skills, logic skills, and a host of other competencies that apply equally well to other academic pursuits.

- ***Improve Math Skills.***
  Math is the universal language that all computational devices speak. In order to code effectively, students learn how to manipulate numbers and other abstract concepts. Coding, as a complement to standard math instruction, can greatly amplify a child's ability to think mathematically and computationally.

- ***Think Computationally.***
  Computers are dumb. They can only do what a programmer tells them to do. In order for kids to learn to do this effectively, they need to make a plan, write code to execute that plan, and then refine their code when it fails, or when it underperforms. Over time simple programs become elaborate instruction chains, with branching logic and contingent decision points. This sort of methodical thinking is useful everywhere in life.

WRAPPING IT UP

In summary, programming a computer is an important skill in world that runs on technology. Computational Thinking is even more important!

UCode builds Computational Thinking foundations by teaching coding. We teach in Python, which is a high-level language very accessible to early learners. We use a Block interface that we designed and that allows kids to code as soon as they have basic math skills. We use robots as a teachig platform which helps students with understanding the relationship between the code they are writing and the output (what the robot does). We use a "challenge based" curriiculum that allows studenst to learn at their own optimal pace and we measure learning progress using a framework developed by UCode and Cornell University.