



# OPAS Prepare Workgroup

## TeachScheme: Collected Comments

Jo Oshiro– August, 2008  
Updated September 12, 2008

Scheme was brought to my attention by Peter Schmurr of Glencoe High School. I have collected here the substance of several emails and some web research.

- **Teach Scheme FAQ:** <http://www.teach-scheme.org/Notes/scheme-faq.html> -- according to the authors, Scheme can easily be used for graphics and gaming applications, and is in use in professional environments.
- **Teach Scheme Overview:** <http://www.teach-scheme.org/Overview/> -- the curriculum as presented for postsecondary educators combines with Java. The vision is to embed Computer Science in the liberal arts.
- **Teach Scheme Workshops:** <http://www.teach-scheme.org/Workshops/> -- free summer five-day workshops have been offered since 1995 (except for summer 2006) and are open to high school teachers:
  - “We also welcome high school teachers who cover at least one of computer science, mathematics, and the physical sciences. *We ask that you first contact your school to determine whether they will allow you to make curricular changes based on the summer course. You should find out what obstacles you will face from your school if you decide to adopt this curriculum*”

**Peter Schmurr, Technology Coordinator/Instructor, Glencoe High School, Hillsboro, Oregon**  
[schmurrp@hsd.k12.or.us](mailto:schmurrp@hsd.k12.or.us) 503-844-1900 X 3694

*Update September 2008: Peter Schmurr has joined the ETIC CS Task Force*

Peter has been teaching math, science and computer programming at Glencoe High School in the Hillsboro School District for 15 years. He has attended three PLTW Summer Training Institutes and has been teaching PLTW courses for the last two years. His 08-09 teaching assignment includes PLTW courses Principles of Engineering, Digital Electronics and Engineering Design and Development. He was a presenter for the Superintendent Summer Institute session “Project Lead the Way: Practicing Literacy, Numeracy, and Essential Skills”. During the course of the day, I asked about cs4hs, as he had just returned from Carnegie Mellon. His response was that he likes Scheme (*Jo’s insertions are italicized*):

Using the Teach-Scheme approach, Glencoe has had 50 intro programming students two years in a row. The course feeds into an AP Computer Science course. Peter would be happy to lead a workshop for interested teachers sometime or “we could bring one of the principal authors out.”

<http://www.teach-scheme.org/> is the educational outreach of the organization (*originating at Brown and Rice*). The textbook and IDE are online for free at <http://htdp.org>. The textbook is undergoing revision and the new edition (not yet online) begins with graphics libraries so students can begin programming interesting visuals.

The success of this approach is due in large part to a distillation of best practices summed up in what the authors (*Matthias Felleison, Robert Bruce Findler, Matthew Flatt, and Shriram Krishnamurthi*) call a "design recipe". The early design recipe has the following steps, and an example is given on the next page.

1. contract
2. header
3. purpose statement
4. test cases
5. body

Students have early success with programming and progress rapidly to complex problems.

When they get stuck the teacher simply walks them through the design recipe again to find where things went wrong.

As data types become more complex, additional steps are added to the recipe.

The design recipe in action for a simple function follows. Lines preceded by semicolons are comments; the words in brackets are not part of the usual comments and are included to identify the parts of the recipe (*translated into table format for what Jo hopes is improved clarity on paper*)

<code>;add1:num - &gt; num</code>	[this is the contract. It identifies the number and data type of the input variables and the data type of the output]
<code>; consumes a number and returns the number plus two</code>	[this is the purpose statement. It clearly states what the function will be designed to do]
<code>;(define (add2 n).....)</code>	[ this is the header. in it we give names to the input variables]
<code>(check-expect (add2 0) 2)</code>	[these are test cases. They are written <b>before</b> defining the body of the function. This forces students to think through the algorithm or calculation before writing code]
<code>(check-expect (add2 10) 12)</code>	Test case
<code>(check-expect (add2 -5) -3)</code>	Test case
<code>(define(add2 n)   (+ n 2))</code>	[ this is the body of the function. It is the <b>last step</b> in the design process.]

*Michal Young, Associate Professor of Computer Science, University of Oregon*

[Michal@cs.uoregon.edu](mailto:Michal@cs.uoregon.edu) 541-346-4140

I've heard of teach-scheme --- it came out of a group at Rice University. I'm organizing a conference right now with Shriram Krishnamurthi, one of the TeachScheme principals (mostly ex-students of Matthias Felleisen when he was at Rice).

I don't know a whole lot about the curriculum and materials, but I can easily imagine that Scheme would appeal to someone from a math background. Scheme is closer than most programming languages to lambda calculus, a way of expressing what programs mean and do mathematically.

Joyce Creswell of Saturday Academy mentioned to me once that her son took a course in Scheme (in high school, or maybe in an after-school program), and Joyce felt that using a non-mainstream language had a useful leveling function --- students who had the logic and math ability but had never programmed before (i.e., girls) were on an even footing with the boys who had been

teaching themselves a little programming.

(Jo:)I realize function-based languages are not particularly mainstream, but maybe there are things about this program/language that are developmentally appropriate. The teacher I talked to liked it because within the IDE you can set a language level so that error messages etc. are appropriate to the learning stage of the programmer (analogous, I think, to Pilot within Robolab, which I had success with for 9 year old girls, pushing them up the levels without scaring them off too soon).

You might also recall Logo, which is really Lisp with a slightly different syntax and turtle graphics. It was pretty successful in its time, although turtle graphics wouldn't impress anyone today.

There are two points of view about choosing an introductory programming language. One (which I personally find attractive) is that it really doesn't matter whether the introductory language is "mainstream", and that it should basically not get in the way of the core concepts. Especially it should not have a lot of fancy syntax, because students (and teachers) can think that learning to program consists of learning all those syntactic details, rather than understanding the underlying concepts. If you subscribe to this philosophy, you may be willing to use one programming language in an introductory course and switch students to a more mainstream language after half a year or a year.

The second point of view is that learning a programming language is hard enough for students new to computer science, and asking them to switch languages before they have become really expert programmers is just asking too much. It looks to those of us who know lots of programming languages that it shouldn't be a big deal, but for beginning students it is. If you subscribe to this philosophy (and I have to admit it has merit), then you may be more inclined to start right out with a language like Java or at least like Python.

Scheme is a good fit for the first of the two views above. There is a bare minimum of syntax (it's hard to imagine getting by with less) and a handful of very powerful building blocks for composing all sorts of program frameworks. The book "Structure and Interpretation of Computer Programs", by Abelson and Sussman, is an amazing presentation of many deep and cool concepts, all built up from the basic building blocks of Scheme. It serves as an introductory textbook at MIT, but I don't think it could be used that way anywhere else.

I have mixed feelings. I'm an old Lisper, and as I mentioned I really like the idea of skipping by all those syntactic details and getting right to the essential concepts ... but I do worry about credibility with students. It might actually work better at the high school level than it does at the college level, because students don't necessarily have the expectation that they are going to be able to immediately put their new knowledge to work on some real-world project.

Bottom line for me: I really think we need math teachers on our side, and if scheme is attractive to math teachers, I'm all for it.

**Bruce Schafer, Executive Director of ETIC, former founder of PC-Kwik**  
[Bruce\\_Schafer@ous.edu](mailto:Bruce_Schafer@ous.edu) 503.821.1132

I don't know much about Scheme but I know that my friends at Galois (including Laura McKinney, the "poster girl" of the GetReal home page) are big fans of Scheme. If we ever decided to go that way I expect they would be willing to help

**Chris Brooks, Chairman of the Board, Techstart Education Foundation**  
[brookscl@gmail.com](mailto:brookscl@gmail.com)

Michal has much more experience with this than I, though I did study both Scheme and Lisp in college. I think there are positives when looking to the math side (recruiting teachers if not students), but I have similar concerns about it being so non-mainstream in terms of use and application. Clearly it could be a great choice for those that have already selected CS as a topic of major interest, but I worry about it being any sort of hook to pull new students in. The hook is important for high school where we are trying to awaken a (hopefully) latent interest. I should probably spend some time with the tools and materials before commenting much more... things have probably changed in the 20 years or so since I was using it.

**Wells Matthews, Software Engineer, Microsystems Engineering Inc.**  
(Jo's husband; graduated from the OMSE program 4 years ago)

Wells tutored our de facto niece in an introductory programming course using Scheme when she was a freshman at the Art Institute of Portland. Although he finds many things to like about the language, he feels it is fundamentally unfair to teach only an intro course in a particular language.

Preface: I'm a professional programmer/SW engineer for 30 yrs. I've been through 2 master programs: MS CS, and MS Software Engineering. I am NOT a cs/programming educator, but as Jo mentioned I tutored our defacto-niece in her 'Intro to Programming' college course that was done in Scheme. My take on using Scheme for an 'Intro to Computer Science' course:

1. As an old Lisper, I like Scheme quite a lot, and think its a fine choice to teach computer science fundamentals. But I wouldn't choose it as an "intro course" language.
2. I feel that a language used in an "intro course" should use the same programming paradigm as most used languages used by computer science majors. This means somewhere between pure imperative (like c, pascal, ...) and pure OOP languages (smalltalk, java, C#, ...).
3. Languages with a different programming paradigm I would leave for later 'advanced' classes. Say use scheme for 'Functional programmer', prolog for 'logic programming' and so on.
4. I concur with other commenters that an initial language should be:
  - relatively clean/small syntax and semantics
  - easy to get started in
  - rich libraries to help the student 'grow'

Given my criteria above, my highest choice today is probably python, with honorary mention to Java, C#, and while hesitant to admit it, Visual Basic.

***Don Kirkwood, AP CS Teacher at North Salem High School also sits on the OPAS Initiative's Succeed Workgroup  
Kirkwood\_don@salkeiz.k12.or.us***

Don's comments to Jo are that the group championing Scheme had a history of less-than-conciliatory behavior that has apparently been corrected in recent years. Don also showed Jo some notebooks of commercially available curricula that he considers to be of good quality. Don's most recent innovation in his teaching leverage the North Salem "A/B" schedule: he couples the subject matter in his freshman Honors Geometry with his Intro CS class. Girls respond particularly well to this integrated approach. For more information on Don's program see "A Community Conversation: Diversity in the CS Classroom" at [http://opas.ous.edu//Workgroups2007/Succeed/Community\\_Conversation\\_Archive.html](http://opas.ous.edu//Workgroups2007/Succeed/Community_Conversation_Archive.html).

***Additional emails are appended below in the pdf version of the file. Background on the participants:***

***Walt Mayberry, one of the founders of Sequent Computers, former high school computer science instructor, entrepreneur and current chair of the OPAS Prepare Workgroup  
wmayberry@comcast.net***

***Other links relevant to the larger conversation on CS curricula***

- **ACM's 2<sup>nd</sup> edition of the K12 Model Curriculum** - <http://csta.acm.org/Curriculum/sub/ACMK12CSModel.html>  
Not a turnkey curricular package, survey comments show it to be valuable for writing standards, curriculum and classes in the US and abroad.
- **CS Unplugged** - <http://csunplugged.org/> - "Computer Science Unplugged is a series of learning activities that reveals a little-known secret: computer science isn't really about computers at all. Unplugged teaches principles of computer science such as [binary numbers](#), [algorithms](#) and [data compression](#) through games and puzzles that use cards, string, crayons and lots of running around. The great ideas behind computer science are brought into the open when we leave behind the details of particular computer systems.

Unplugged is suitable for people of all ages, from elementary school to university, and from many countries and backgrounds. Unplugged has been used around the world for over fifteen years, in classrooms, science centers, homes, and even for holiday events in a park!"

- **Cs4hs** from Carnegie-Mellon University – answers from Helene at UW to questions posed by Ron Tenison, President of OCSTA - [http://www.oregonetic.org/meetings/CStaskforce/Helen\\_Answers.pdf](http://www.oregonetic.org/meetings/CStaskforce/Helen_Answers.pdf) - cs4hs is aimed more at integrating CS concepts and lessons throughout math and science curricula than at creating a separate strand or class, and involves in part demonstrating the use of CS in other disciplines. One of the centers of activity and summer workshops for teachers is at the University of Washington. This material has been under discussion at the ETIC CS Task Force meetings.

**Oshiro, Jo**

---

**From:** Schmurr, Peter [schmurrp@hsd.k12.or.us]  
**Sent:** Friday, August 08, 2008 6:00 PM  
**To:** Oshiro, Jo  
**Subject:** RE: The Teach Scheme, Reach Java CS project/curriculum

Jo,

Thank you for bringing this to the attention of OPAS. I'm glad that TeachScheme! has become part of the discussion. I am happy to answer questions that members may have about the program and perhaps post up examples of cool games that my students have written in Dr. Scheme.

Take care,

Peter

---

**From:** Oshiro, Jo [mailto:Jo\_Oshiro@ous.edu]  
**Sent:** Fri 8/8/2008 4:18 PM  
**To:** Schafer, Bruce; brookscl@gmail.com; Schmurr, Peter; tenison@verizon.net; Walt Mayberry; Michal Young; wells.matthews@biotronik.com; Jay Bockelman; Dick Knight  
**Subject:** The Teach Scheme, Reach Java CS project/curriculum

Thanks to those of you who responded to my questions. I'm putting out these collated responses to the original respondents and others whom I judge to be interested.

Jo Oshiro  
Program Coordinator  
OPAS - Oregon Pre-Engineering and Applied Sciences Initiative  
OUS, Industry Affairs - Capital Center #1065, 18640 NW Walker Rd, Beaverton 97006  
v 503.821.1139  
f 503.690.8096  
[jo\\_oshiro@ous.edu](mailto:jo_oshiro@ous.edu) - <http://opas.ous.edu/>

## Oshiro, Jo

---

**From:** Walt Mayberry [wmayberry@comcast.net]  
**Sent:** Sunday, August 10, 2008 7:16 PM  
**To:** Oshiro, Jo; Schafer, Bruce; brookscl@gmail.com; schmurrp@hsd.k12.or.us; tenison@verizon.net; Michal Young; wells.matthews@biotronik.com; Jay Bockelman; Dick Knight  
**Subject:** Re: The Teach Scheme, Reach Java CS project/curriculum

Hi Jo,

I just read through the comments from the other reviewers on Pete Schmurr's use of TeachScheme. I did look at this a few years ago as an option in my CS curriculum.

I taught CS and other computer subjects (computer systems, computer networks, computer applications) at Lake Oswego HS and Lakeridge HS for three years - 2000 - 2003. LOSD has a very academic, non-vocational, focus, so this all had to be college preparatory - so no Cisco Academy, STRUT, etc. I went through all the struggles to define an appropriate CS curriculum that I still see others going through. I was never fully happy with the result, and though I left teaching, I have been looking for that perfect curriculum ever since. Unfortunately, and to a degree unbelievably, no one else seems to have found it, either.

LOSD was strong into AP classes, so I chose to go with AP-CS. It wasn't perfect, but it was close enough and was a full package deal for me as a teacher - much as we now see PLTW - curriculum, text books, teacher training, assessment, nationwide recognition, discussion forums, etc. They sent me back to Duke Univ to do a weeklong workshop with the chief reader. It was great, and inspirational. And I joined the AP-CS discussion forum, which turned out to be the best teacher support group I ever connected with. This was the year before AP-CS switched from being C-based to Java-based. I went ahead and started with Java to avoid having to switch the second year (buy new text books, etc). On the whole, I think this was the right choice. It gave students the right preparation and credentials to enter a college CS program, and prepared them for the rigor to expect there. On the down side, I don't think very many of my students actually went on to study CS in college. This was in the middle of the Internet-bubble crash, and offshoring, so it wasn't thought of as a very positive field. I do know one ex-student did create a software business on the side while in college. The other downside is that this was not an entry level class. There was not enough time in a year to take students who knew nothing about CS and to prepare them for the AP-CS exam by early May. So, I had semester long Intro to Programming and Intermediate Programming classes leading up to this. This funnel, and relatively small size of the schools (~1200) made it hard to get the minimum number of students into the class to justify it (30 students). In the beginning the size requirement was waived and I held the class with 18 students, but I could never get 30 and it was not to be held in the 4th year (they canceled the whole program, so this was a moot point in the end).

For the introductory CS curriculum, I wanted to use a language what was more approachable than Java and with which you could get more graphical glitz and build real applications (eg games) with less overhead. I chose Visual Basic. Maybe not perfect academically, but it did attract students to the classes because they could do real things soon into the class. I broke this into two separate semester classes - one called Introductory and one called Intermediate. Even with attempts to make this a "fun" programming class, I often ended up with serious misfits. Many students thought that because they were great at playing computer games, they would be good at creating them. The real pre-requisite was "good at algebra". Others just weren't ready to do the work - one was outraged that I assigned homework - "I thought this was a elective like 'cooking', you mean I have to study for this class!" In the end, I was able to fill about 2 Intro and 1 Intermediate classes per semester, which I viewed as OK success.

In the end, the program was canceled at both high schools and the matching junior high schools, largely due to a budget crunch that caused a focus back on core academics. And for electives, they preferred to put the money into classes that could pull 50 students (band, choir) rather than ones that struggled to get 30 (and required an expensive PC lab).

In retrospect, I looked at some alternative introductory approaches, like the Alice graphically programming environment, KarlJ Robot, TeachScheme, and Python. In the end, I decided that wasn't the real problem. I think

the problem was more the focus on college CS preparation. These schools (and most in Oregon) were just too small and just didn't have the demographics to get critical mass (multiple classes of 30-per-day) in this area. Perhaps a better approach would be a more general reaching out to teach problem solving through programming, which can be of interest and utility to all students in any discipline (like CS4HS?). If this is the goal, you probably want to teach in a more "mainstream" language like Visual Basic or Python, rather than a more conceptually pure, but academically focused language like Scheme. If the focus is to be college CS preparation, and there isn't a concern about filling seats, I think TeachScheme is a better starting point, from what I have learned.

Walt

----- Original Message -----

**From:** [Oshiro, Jo](#)

**To:** [Schafer, Bruce](#) ; [brookscl@gmail.com](mailto:brookscl@gmail.com) ; [schmurrp@hsd.k12.or.us](mailto:schmurrp@hsd.k12.or.us) ; [tenison@verizon.net](mailto:tenison@verizon.net) ; [Walt Mayberry](#) ; [Michal Young](#) ; [wells.matthews@biotronik.com](mailto:wells.matthews@biotronik.com) ; [Jay Bockelman](#) ; [Dick Knight](#)

**Sent:** Friday, August 08, 2008 4:18 PM

**Subject:** The Teach Scheme, Reach Java CS project/curriculum

Thanks to those of you who responded to my questions. I'm putting out these collated responses to the original respondents and others whom I judge to be interested.

Jo Oshiro

Program Coordinator

OPAS - Oregon Pre-Engineering and Applied Sciences Initiative

OUS, Industry Affairs - Capital Center #1065, 18640 NW Walker Rd, Beaverton 97006

v 503.821.1139

f 503.690.8096

[jo\\_oshiro@ous.edu](mailto:jo_oshiro@ous.edu) - <http://opas.ous.edu//>

**Oshiro, Jo**

---

**From:** Michal Young [michal@cs.uoregon.edu]  
**Sent:** Sunday, August 10, 2008 8:53 PM  
**To:** Walt Mayberry  
**Cc:** Oshiro, Jo; Schafer, Bruce; brookscl@gmail.com; schmurrp@hsd.k12.or.us; tenison@verizon.net; wells.matthews@biotronik.com; Jay Bockelman; Dick Knight  
**Subject:** Re: The Teach Scheme, Reach Java CS project/curriculum

Walt,

Thanks for the insightful comments, including the history lesson.

[The real pre-requisite was "good at algebra".](#)

In practice this tends to be a very good rule of thumb at the college level as well. Our course catalog says something silly about programming experience, but when advising incoming students we ask them about the math they've taken, and whether they like it. If they can do math and especially if they can do it *and* they like it, then I have no qualms sending them into a course that supposedly requires some programming experience. I think most of my colleagues do the same. Since we're looking at recent high school graduates, "good at math" generally means having done well in at least elementary functions (pre-calc), but in truth it's only the algebra that matters.\* It's just that if algebra is the last course a high school student took, it means they don't like math or they're scared of it. (And we still try to help those students enter the program, but they don't start in the same place.)

Perhaps a better approach would be a more general reaching out to teach problem solving through programming, which can be of interest and utility to all students in any discipline (like CS4HS?).

I would personally be delighted with broad exposure to the problem solving methods used in computer science for students in all disciplines ... and as for preparing students for college CS, the main thing I'd like is for students who ought to be interested in CS to discover it. They can program a TI calculator for all I care, if that catches their fancy and helps them make a good judgment about the field. (That might include helping some students understand that CS isn't what they want to do ... like the students who were confused about the difference between playing video games and building them.) I'm even open to approaches that don't teach any programming at all, although I'm skeptical that a student can really grok what CS is without writing programs of some kind.

Of course it's cool if students get a head start on the actual college-level content early. But really ... I don't care. If a kid can think clearly, can move between abstract and concrete (like solving an algebra story problem), and is willing to challenge him or herself to think computationally, then we're ready to rock and roll. I think parents care a lot about AP credit and having a head-start on real college classes. I think college educators care much more about kids who can think. But, we've got to get them in the door. A kid with excellent preparation to do college-level computer science, but who never considers taking a college-level computer science course, is a terrible missed opportunity.

If this is the goal, you probably want to teach in a more "mainstream" language like Visual Basic or Python, rather than a more conceptually pure, but academically focused language like Scheme. If the focus is to be college CS preparation, and there isn't a concern about filling seats, I think TeachScheme is a better starting point, from what I have learned.

What I've written above sounds like a contradiction to what I wrote before about liking scheme as an instructional language (though I also expressed some reservations). What I like about Scheme is actually consistent with my comment about wanting thinking and abstraction skills first ... but I accept Walt's comment about the need to fill seats, and the last thing I would want to do is make CS seem unexciting to students who ought to be knocking our doors down. I'm completely fine with Visual Basic or Python or almost anything else ... tinkertoys for all I care ... as long as it doesn't devolve into a course on the syntax of a language.

--Michal

\*One exception to my comment above that the real math content we use is just algebra, although we want students to have continued in math beyond algebra for "mathematical maturity". The other specific bit of math that helps a lot is proof by induction. If a student really groks proof by induction, then recursion is going to be a piece of cake when they see it, and these days they see it in the first programming course. Being really comfortable with mathematical proofs is indispensable in the second year courses, but CS departments require a discrete mathematics sequence for that.

**Oshiro, Jo**

---

**From:** Schmurr, Peter [schmurrp@hsd.k12.or.us]  
**Sent:** Wednesday, August 13, 2008 9:57 AM  
**To:** wells.matthews@biotronik.com; Oshiro, Jo  
**Cc:** brookscl@gmail.com; Schafer, Bruce; Jay Bockelman; Dick Knight; Michal Young; tenison@verizon.net; Walt Mayberry  
**Subject:** RE: The Teach Scheme, Reach Java CS project/curriculum

*"I feel that a language used in an "intro course" should use the same programming paradigm as most used languages used by computer science majors. This means somewhere between pure imperative (like c, pascal, ...) and pure OOP languages (smalltalk, java, C#, ...)"*

The authors originally called the program **TeachScheme!** with an exclamation mark as in teach scheme (**not**). The underlying pun was intended to emphasize that the curriculum was not about teaching Scheme . The real strength of the curriculum is that by the time students make the transition to a difficult syntax laden language like Java they have already learned a disciplined approach to program design. In so doing they had exposure to a different paradigm along the way. It has been my experience that the skills learned carry over very well into the next language. For students who do go on to take AP Computer Science, many score 4s and 5s on the exam.

It is this focus on design discipline and computer programming as problem solving that has been missing from other approaches I have tried in the past.

Whether or not students go on to additional programming courses, they all take away a set of problem solving skills that are transferable to other disciplines.

Regards,

Peter

Peter Schmurr

Technology Coordinator/Instructor  
 Glencoe High School  
 503-844-1900 X 3694

---

**From:** wells.matthews@biotronik.com [mailto:wells.matthews@biotronik.com]  
**Sent:** Tue 8/12/2008 7:21 PM  
**To:** Oshiro, Jo  
**Cc:** brookscl@gmail.com; Schafer, Bruce; Jay Bockelman; Dick Knight; Michal Young; Schmurr, Peter; tenison@verizon.net; Walt Mayberry; wells.matthews@biotronik.com  
**Subject:** Re: The Teach Scheme, Reach Java CS project/curriculum

preface: I'm Jo's husband, a professional programmer/SW engineer for 30 yrs. I've been thru 2 master programs:

9/12/2008

MS CS, and MS Software Eng. I am NOT a cs/programming educator, but as Jo mentioned I tutored our defacto-niece in her 'intro to programming' college course that was done in scheme.

my take on using Scheme for an 'intro to computer science' course:

1. as an old Lisper, I like scheme quite a lot, and think its a fine choice to teach computer science fundamentals. But I wouldn't choose it as an "intro course" language.
2. I feel that a language used in an "intro course" should use the same programming paradigm as most used languages used by computer science majors. This means somewhere between pure imperative (like c, pascal, ...) and pure OOP languages (smalltalk, java, C#, ...).
3. Languages with a different programming paradigm I would leave for later 'advanced' classes. Say use scheme for 'Functional programmer', prolog for 'logic programming' and so on.
4. I concur with other commenters that an initial language should be:
  - relatively clean/small syntax and semantics
  - easy to get started in
  - rich libraries to help the student 'grow'

Given my criteria above, my highest choice today is probably python, with honorary mention to Java, C#, and while hesitant to admit it, Visual Basic.

-- wells

Wells Matthews  
 Staff SW Engineer  
 Micro Systems Engineering, Inc.  
 6024 S.W Jean Road  
 Lake Oswego, OR 97035  
 503-635-4016 x1269  
 wells.matthews@biotronik.com

"Oshiro, Jo" <Jo\_Oshiro@ous.edu>

08/08/2008 04:18 PM

To "Schafer, Bruce" <Bruce\_Schafer@ous.edu>, <brooksci@gmail.com>, <schmurrp@hsd.k12.or.us>, <tenison@verizon.net>, "Walt Mayberry" <wmayberry@comcast.net>, "Michal Young" <michal@cs.uoregon.edu>, <wells.matthews@biotronik.com>, "Jay Bockelman" <Jay.Bockelman@oit.edu>, "Dick Knight" <knight@hevanet.com>

cc

Subject The Teach Scheme, Reach Java CS project/curriculum

Thanks to those of you who responded to my questions. I'm putting out these collated responses to the original respondents and others whom I judge to be interested.

Jo Oshiro  
 Program Coordinator  
 OPAS - Oregon Pre-Engineering and Applied Sciences Initiative  
 OUS, Industry Affairs - Capital Center #1065, 18640 NW Walker Rd, Beaverton 97006  
 v 503.821.1139

9/12/2008

f 503.690.8096

[jo\\_oshiro@ous.edu](mailto:jo_oshiro@ous.edu) - <http://opas.ous.edu//>