Picking a Beginning Programming language

There is a companion spreadsheet [http://bit.ly/NETA19-python1] that has been filled out by the members of the Nebraska chapter of CSTA for a variety of programming language based on the criteria below. You might be interested in what they thought, or you might want to copy the spreadsheet and fill it out for your preferred language. You can set the weight for each criteria below.

When rating languages, let's define 0, 5, & 10 as below and use intermediate values as they make sense.

10: Really excellent at this criteria -one of the main reasons to use this language
5: Neither a plus or minus
0: A definite drawback for this language

Questions to ask in evaluating programming languages in an academic setting.

Questions pertaining to the language itself:

Is the language available for the platform(s) at my school? You will probably not have the resources to change platforms just to use a particular programming language, but you might be able to dual-boot to accommodate a certain language. (C# on OS X or Objective C on windows)

Is the language available for the platforms that my students are likely to have at home? This is more problematic as you don’t have any control over it.

How much does the language cost? This could have bearings on both school and home use.

Is the language proprietary? A language that is only available from one vendor might be less likely to be relevant over a long period of time.

Questions about usage of the language

Is the language used in industry? How widely? Prominent companies? This might not be a big concern in some situations.

Is the user community helpful? This could also include customer support for proprietary languages. Are they a source for answers for you? For your students?

Are community code examples likely to teach good coding techniques? When your
advanced students look for more advanced things that they can do, how likely are they to find things that promote good programming?

Is it used to teach intro courses in higher education? If no one in higher ed. uses it, is there a reason? If some use it, are there resources that they make available that you can use as well? Online lectures? Homework ideas? Textbooks?

Is the language likely to be relevant in the future? This is pretty subjective, but here are some websites: http://www.langpop.com/
http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Questions about teaching with the language:

Can you teach imperative programming? Traditional control structures, subroutines (“normal” programming).

Can you teach functional programming? Languages like Lisp, Scheme, OCaml, Haskell.

Can you teach object oriented programming? Pure OO languages like Smalltalk and Ruby or languages that also support procedural programming like most other languages these days.

Can you teach "objects first", “objects early”, “objects late”? Some languages force you to do one of these, others give you some flexibility to make decisions for pedagogical reasons.

The language is small enough for students to keep it in their head? Can students write the programs that you want them to write without a reference book? this is also true if there is additional complexity that beginners can safely ignore.

How much "hand waving" is required at the start? How much do you have to explain to show a first program? How often do you say, “We will learn that later”?

What do “Hello World” and “Sum of two numbers” look like? Let’s look at two very small programs and see how hard they are to explain to beginning programmers.

Is the language applicable to your grade level? Logo might not be a good choice for your Jr/Sr level programming course. Or maybe it is.

Are there easy ways to introduce the language (turtle graphics)? Is there a subset of the language that you can teach to get across basic programming concepts?

Programming Topics

How easy is it to cover topics like: iteration (loops), conditionals, recursion, arrays, multidimensional arrays, pointers, algorithmic complexity, memory management

Questions about using the language.

Does the language invite code reuse? Is it easy for beginning programmers to reuse code in other programs? Are there mechanisms in place for importing procedures from external files?

Are error messages useful for debugging? Can the error messages be used to pinpoint the most likely cause of the problem? Do they help explain the thing that the programmer did that the computer didn’t expect? Are they approachable by beginners?

Are programs "readable"? How hard is it for a beginning programmer to read through a program and tell approximately what is happening? What is the ratio of English to Symbols? “Perl is executable line noise”, “Python is executable pseudocode” Most languages fall somewhere in between and usually based more on the programmer than the language.

Is documentation well supported? Is commenting code easy? Are there any other mechanisms that support well documented code?

Concerning IDEs: Are there IDEs available? Are there free IDEs available? Do you NEED an IDE? There is a school of thought that an IDE just gets in the way of a beginning programmer. It is just another thing to learn at the start.

Are there debuggers available? Can you step through a program? Can you set breakpoints? Is there a way to examine variable values?

Are there profilers available? Are there tools included or available to see how efficient your code is?

Are there useful libraries available? Can you parse HTML? connect to Twitter or the Google search API? Is there a GUI that is easy to implement?

Are there high level data structures built in? This may or may not be a plus for you depending on what you’re teaching.
When can students write programs they can "feel good about"? How long will it take to write a program that students consider non-trivial? Maybe that’s defined as one that by writing a program they saved time over doing the same task by hand?

When can students write programs that would impress their friends? This is probably harder to do. Can you write a program that they would recognize as an "actual" program? Is there some other final product that this language offers instead?

**Technical Questions:**

**Interpreted or compiled?** Some people feel that one is better than the other for teaching? Do you learn more about how a computer works by compiling? Is the quick feedback nature of an interpreted language better?

**Static or dynamically bound variables?**

**Strong or weakly typed variables?**

**Considerations for future CS coursework**

**How easy will it be to learn other common languages?** Are there things about this language that will make other languages harder to learn? Little idiosyncrasies that students might not understand are not common across other languages?

**Does it lend itself to good habits like code reusability?** This one is pretty nebulous. What are other good coding habits? Would anyone rate a language low in this?