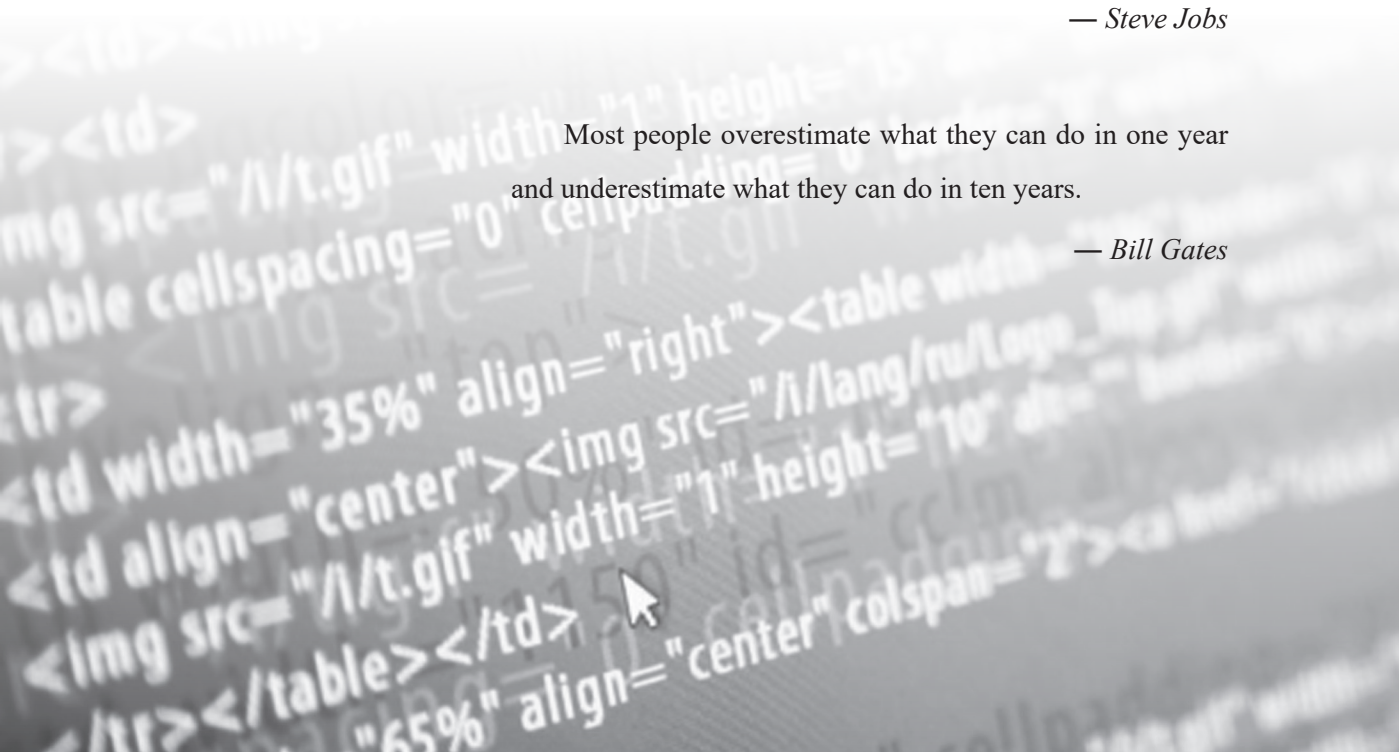
Unit 1
Finding My Niche

Have the courage to follow your heart and intuition. They somehow know what you truly want to become.

— *Steve Jobs*

Most people overestimate what they can do in one year and underestimate what they can do in ten years.

— *Bill Gates*



Preview

INFORMATION TECHNOLOGY continues to change the way we live, play, and do business. The dominance of the IT job market is due in part to numerous factors, including the prolific growth of the Internet and e-commerce, lower hardware and software prices that allow more businesses to upgrade their technology, increased demand for information security specialists spurred by the escalating frequency and sophistication of cyber-crimes, the advent of smarter applications that enable companies to analyze data and develop unprecedented business intelligence, and the dawn of the mobile computing era. However, despite rapid growth and increased opportunity, simply showing up will not guarantee success. The IT job market will continue to get more competitive as people go where the money and jobs are. This is why it's important to clearly identify your career objectives and develop a learning plan with the necessary skills, computer training and IT certifications to build a competitive edge and achieve your goals.

Section A

Pre-reading Activities

1. Check (√) the statements which you think are important for programmers.

☐ a) Math skills
☐ b) Good design
☐ c) Attention to details
☐ d) Patience
☐ e) Self-learning skills
☐ f) Logical, precise, rigorous thinking
☐ g) Problem-solving skills
☐ h) Good communication skills

2. Work in pairs and discuss the following questions.

a) What job in IT field attracts you most? Why?

b) What should you do before you choose your career?

c) Why do IT majors need to learn math?

Text A

What Does It Take to Be a Programmer?

1. Many people want to know if they **have what it takes** to be a good programmer. There's no simple, check-these-boxes answer to the question, but there are some helpful **traits** that you may have or that you can develop.

2. Do I need to know Math? First, let me **eliminate** a **myth**: some people think that math skills are important, but I've seen great mathematicians who are **mediocre** programmers and lots of great programmers who are certainly not mathematicians (and probably never expected to be).

3. Programming is more of a designer's task—to be a good programmer, **having an eye for** style and good design is extremely helpful. I don't mean the type of style that **governs** where you put pieces of **syntax**. For instance, in C, there are several places where you can put **curly braces**[1] to surround blocks of code, and while there are heated debates about whether

```
if (...) {
}
```

is better than

```
if (...)
{
}
```

4. These are small points of little consequence, and as long as you are consistent, this trait will eventually come naturally. What I really mean by style is that you have to **have a good sense for discriminating** between "good" and "bad" approaches to attacking problems.

5. Design is important. When you first sit down to write a program, you probably don't know exactly what it should do (or how to do it). If you**'re disciplined about** it, you'll take some time to plan things out on paper and figure out more or less what you'd like your program to do. That's great, but it won't substitute for having actually used the program and noticed that, yes, it would be fantastic to add this one little feature here.

6. The secret is that adding little features can be very hard! This seems surprising to someone who's never programmed before: all you need to do is have it **print**[2] this one piece of data, or take this one type of input, etc. The problem is that inside the program, the **architecture** might not be designed to support that kind of information. For instance, let's say that you wanted to move a button from one place to another on a simple **graphical user interface**[3]. If the program has been well-designed, this shouldn't be too much of a problem, but if it hasn't, well, consider this possibility: the position of the button is governed by its location in **pixels**. All button locations are **hard-coded**[4] into the program. Now, if you move one button, you may have to go back and change where every single button is located both in the **routine** to draw the buttons and in the routine to accept the input. This can be quite a **hassle**!

7. Clearly, you want some way of having a notion of button positions that isn't quite so hard to change. But if you started out your program and didn't consider that it would be nice to be able

to move the buttons around, you have to go back and change possibly 20 or more lines of code (say, two for each button) just to move one of ten buttons. And if you make a mistake with one button, you're likely to see unforeseen results whose cause is hard to discover.

8. This kind of program design is **brittle**: it can work at first, but when you need to change something, it's not flexible. Each button depends on every other button and relies on the programmer to make the changes. A much better approach would be one in which the positions of buttons when they're drawn and when they're clicked on are linked—changing one wouldn't mean you have to change the other.

9. The more willing you are to put in the **up-front** thinking before designing your program, the easier you will find the actual writing of code. This is not to say that when you're first learning you shouldn't just write some simple programs without worrying too much about these issues. But you should be prepared to pay attention to these things and what problems your first programs did have.

10. The second trait that you need is patience. At some point in your programming career, you will certainly make small mistakes that cost you hours of **debugging** only to realize that you were misspelling a **variable** name so the **compiler** thought it was another variable. These things happen even to good programmers—and the better you get as you practice, the more you find that your bugs are interesting—but still hard to find. If you're not willing to patiently work through possible hypotheses and test each one in turn, you're probably going to find programming to be frustrating as much as it is **exhilarating**.

11. If you're looking to eventually have a programming job full-time, you'll want to acquire **exceptional** patience because you'll almost certainly be expected to spend a great deal of time working on **documenting** your code for other programmers and possibly even hunting bugs in someone else's code.

12. The benefit of all of this is that you gain an eye for small details that can have ripple effects and you become much better at the process of asking yourself what could go wrong and how you can test it. Finally, you have a lot of tools at your disposal to help **mitigate** the problems; you can use the compiler to find syntax errors and **debuggers** to find **runtime errors**[5]. Life is not **bleak**: not all of your time will be spent finding bugs!

13. Third, you need to be able to think in a logical, precise, **rigorous** way—you have to be willing and able to specify all of the details in a process and understand exactly what **goes into** it. This can lead to some amazing realizations—for instance, you will understand almost anything better once you've written a program to actually do it. One story goes that a group of programmers discovered a flaw in a state law passed by the **legislature** when trying to program the logic of the law—it turned out two paragraphs made contradictory statements! Nobody noticed until they tried to make it easy enough so that a computer could understand it. It means that you need to have the ability to eventually understand the **entirety** of a process at the level of detail required for a computer to be able to **mechanically** reproduce it.

14. At the same time, you must be capable of **framing** problems the right way and be or become a good problem solver. While your program may need to accomplish a certain task, don't **get caught up in** the first way you tried to solve the problem. For instance, if you need to store 20 phone numbers, it might make more sense to use an **array** than 20 separate variables. Even though you could eventually write the program that way, it would be much better to write it with the array. It would be a shorter program and an easier program to maintain. Often, restating the problem is a good way of **reframing** it. This is a skill you'll learn over time; you don't need to have mastered it before you start programming.

15. If you are **persistent**, willing to pay attention to issues of design and focus on both problem solving and precise solutions to problems, you will go as far as a programmer. If not, a programming career may turn out to be **exhausting** and tedious.

(Adapted from Alex Allain's "What Does It Take to Be a Programmer?" on *Cprogramming.com*, Dec. 2011)

New Words

trait /treɪt/ *n.* [C] element in sb's personality; distinguishing characteristic 人的个性；显著的特点；特征

eliminate /ɪ'lɪmɪneɪt/ *vt.* **~ sb/sth (from sth)** to remove (esp. sb/sth that is not wanted or needed) 消除；清除；排除（尤指不必要或不需要的某人 / 某物）

myth /mɪθ/ *n.* 1. [C] a story from ancient times, especially one that was told to explain natural events or to describe the early history of a people 神话 2. [C] something that many people believe but that does not exist or is false 很多人相信却不存在或不真实的事或想法

mediocre /miːdɪ'əʊkə(r)/ *adj.* not very good; second-rate; moderate; inferior in quality 不太好的；平庸的；第二流的

govern /'gʌvn/ *vt.* 1.(grammar) to require to be in a certain grammatical case, voice, or mood 支配；限定；需要 2. to influence (sth/sb) decisively; determine 支配某事物 / 某人；决定

v. to rule (a country, etc.); control or direct the public affairs of (a city, country, etc.) 统治（国家等）；控制，支配，治理，管辖（城市、国家等的公共事务）

syntax /'sɪntæks/ *n.* 1. [U] (linguistics) (rules for the) arrangement of words into phrases and phrases into sentences 句法；语句结构 2. [U] (computer science) the rules that describe how words and phrases are used in a computer language [计] 语法，一种程序设计语言的拼写和文法

discriminate /dɪ'skrɪmɪneɪt/ *vt.* **~ between A and B**; **~ A from B** to see or make a difference (between two things) 分别，辨别，区分（两事物）

vi. **~ against sb/in favour of sb** to treat (one person or group) worse/better than others 歧视或偏袒（某人或某些人）

architecture /'ɑːkɪtekʧə(r)/ *n.* 1. [C] (computer science) the structure and organization of a computer's hardware or system software [计] 体系结构；架构 2. [U] the discipline dealing with the principles of design and construction and ornamentation of fine buildings 建筑学；设计建造结构的科学

graphical /'græfɪk(ə)l/ *adj.* 1. relating to or presented by a graph 图解的 2. written or drawn or engraved 绘画的；生动的

pixel /'pɪksəl/ *n.* [C] the smallest discrete component of an image or picture on a CRT screen (usually a colored dot) (显示器或电视机图像的）像素

routine /ruː'tiːn/ *n.* 1. [C, U] an unvarying or habitual method or procedure（日常）程序；例行程式 2. [C] a computer program, or part of a program, that performs a specific function [计] 程序

hassle /'hæsl/ *n.* 1. [C, U] (informal) difficulty; trouble 麻烦；困难 2. [C] (informal) disorderly fighting; dispute 激战；争吵

vt. (informal) to annoy continually or chronically 使……烦恼；搅扰

brittle /'brɪtl/ *adj.* 1. hard but easily broken; fragile 硬而易碎的；脆弱的；(fig.) easily damaged; insecure 容易损坏的；不安全的 2. (of a sound) unpleasantly hard and sharp（指声音）尖利的 3. (of a person) lacking in warmth; hard（指人）冷淡的；难相处的

up-front /'ʌpfrʌnt/ *adj.* advance; frank and honest 提前的；预先的；坦率的

debug /ˌdiː'bʌg/ *vt.* 1. (informal) to find and remove defects in (a computer program, machine, etc.) 检测并排除（计算机程序、机器等）中的故障 2. (informal) to find and remove hidden microphones from (a room, house, etc.) 从（房屋等）中找出并去除窃听器

variable /'veərɪəbl/ *adj.* 1. varying; changeable 变化的；可变的；易变的 2. (astronomy) (of a star) periodically varying in brightness（指星星）亮度周期变化的

n. 1. [C] (often pl.) variable thing or quantity 可变的事物；可变的量 2. [C] (technical) a mathematical quantity which can represent several different amounts 变量

compiler /kəm'paɪlə(r)/ *n.* 1. [C] someone who collects different pieces of information to be used in a book, report, or list 编辑者 2. [C] (technical) a set of instructions in a computer that changes a computer language known to the computer user into the form needed by the computer 从高级语言原始码制造程序的程序；编译器

exhilarating /ɪg'zɪləreɪtɪŋ/ *adj.* very exciting and enjoyable 使人高兴的；令人振奋的

exceptional /ɪk'sepʃənl/ *adj.* very unusual; outstanding 异常的；罕见的；特殊的；杰出的；突出的

document /'dɒkjʊmənt/ *n.* [C] paper, form, book, etc. giving information about sth; evidence or proof of sth 文件；公文；文献

vt. 1. to prove or support (sth) with documents 用文件证实或证明（某事） 2. to record the details of an event, a process, etc. 记录；纪实性

地描述

mitigate /'mɪtɪgeɪt/ *vt.* (formal) to make (sth) less severe, violent or painful; moderate 使（某事物）减轻，和缓；节制

debugger /diː'bʌgə/ *n.* [C] a computer program that helps to find and correct mistakes in other programs 调试程序，排错程序（可帮助找出并修正其他程序中的错误）

runtime/'rʌntaɪm/ *n.* [U] the period during which a computer program is executing 运行时

bleak /bliːk/ *adj.* 1. (of a landscape) bare; exposed; wind-swept（指景物）荒凉的，裸露的，光秃秃的 2. (of the weather) cold and dreary（指天气）寒冷的，阴沉的 3. (fig.) not hopeful or encouraging; dismal; gloomy 无望的；阴郁的；黯淡的

rigorous /'rɪgərəs/ *adj.* done carefully and with a lot of attention to detail 严格的；苛刻的；严厉的；严酷的

legislature /'ledʒɪsleɪtʃə/ *n.* [C] (formal) a group of people who have the power to make and change laws 立法机构；立法院；议会

entirety /ɪn'taɪərətɪ/ *n.* [U] the state of being total and complete 全部；完全

mechanically /mɪ'kænɪklɪ/ *adv.* in a mechanical manner; by a mechanism; in a machinelike manner; without feeling 机械方面地；物理上地；机械地

frame /freɪm/ *n.* [C] a structure supporting or containing something 框架

vt. 1. to express (sth) in words; compose or formulate（用文字）表达（某事）；创作；拟定；制定 2. to put or build a frame round (sth)（给某物）镶框；（给某物）做框 3. (fig.) to give more information about the setting of sth, eg. a problem (in order to define, locate, or analyze it) 综合更多信息（以锁定问题）

array /ə'reɪ/ *n.* 1. [C] (computer science) collection of data arranged so that it can be extracted by means of a special program [计] 数组,阵列（可经某种程序取出的一系列数据） 2. [C] impressive display or series 展示；显示；陈列；一系列 3. [U] (formal) clothes; clothing 衣服；服装

vt. [esp. passive] (formal) to place (esp. armed forces, troops, etc.) in battle order 部署（尤指兵力等）

reframe /rɪ'freɪm/ *vt.* 1. to support or enclose (a picture, photograph, etc.) in a new or different frame 再构造（给……装上新框架） 2. to look at, present, or think of (beliefs, ideas, relationships, etc.) in a new or different way 重新审视或思考

persistent /pə'sɪstənt/ *adj.* persevering; never-ceasing; continually recurring to the mind 坚持的；持续的；固执的

exhausting /ɪg'zɔːstɪŋ/ *adj.* having a debilitating effect; producing exhaustion 使筋疲力尽的；使耗尽的

Phrases and Expressions

have what it takes to have the qualities that are needed for success 拥有达到成功所需的品质

have an eye for	to have a taste or an inclination for someone or something 对……有鉴别力
have a good sense for	to have an ability to make reasonable judgements 对……有很好的判别能力
be disciplined about	possessing or indicative of discipline 受过训练的；遵守规则的
go into	to start to be in a particular state or condition 进入……状态
get/be caught up in	to be absorbed or involved in sth 被卷入或陷入某事物中

Terms and Notes

1. curly brace	花括号，大括号
2. print	输出；显示。这是一个基本的编程语言命令。
3. graphical user interface	图形用户界面，指采用图形方式显示的计算机操作环境用户接口
4. hard-coded	硬编码的；写死的。在计算机程序或文本编辑中，指将可变变量用一个固定值来代替的方法。用这种方法编译后，如果以后需要更改此变量将非常困难。
5. runtime error	运行时错误

Structure Analysis of the Text

Part I

The Question About What It Takes to Be a Good Programmer (Para.____)

↓

Part II

The Elimination of a Myth Whether Maths Skills Are Important (Para.____)

↓

Part III

The Helpful Traits of a Successful Programmer (Paras.____)

↓

Good Design (Paras.____)	Patience (Paras.____)	Logical, Precise and Rigorous Thinking (Para.____)	Be Capable of Framing and Solving Problems (Paras.____)

Comprehension of the Text

I. Answer the questions on the content.

1. What are the helpful traits of a successful programmer?
2. What does "having an eye for style and good design" mean in the 3rd paragraph?
3. What does the author really mean by "style"?
4. Why can it be very hard to add little features?
5. If all button locations are hard-coded into the program, what do you have to do when you want to move one button?
6. How to avoid the situation in Question 5?
7. Why is patience so important to a programmer?
8. What's the benefit of patiently documenting your code and debugging?
9. How did the group of programmers find the flaw in the state law?
10. If you need to store 20 phone numbers, will it be better to use an array or 20 separate variables? Why?

II. Write T (true) or F (false) for the following statements according to the passage.

1. To be a programmer, you must be very good at math. ( )
2. Where to put pieces of syntax shows whether the programmer has a good sense of good design. ( )
3. If you plan out everything, you will not need to make any change. ( )
4. Good designer should take some time to plan things out on paper. ( )
5. If the program has been well-designed, moving a button may result in a big problem. ( )
6. Compilers can be used to find syntax errors. ( )
7. Up-front thinking before designing your program makes code writing easier. ( )
8. If you are a full-time programmer, you will hardly spend any time hunting bugs in someone else's code. ( )
9. Good programmers need to be able to think in a logical, precise, rigorous way. ( )
10. Restating and reframing a problem is a skill you need to possess before you become a programmer. ( )

III. Fill in the blanks with the information given in the text.

1. It is extremely helpful for a good programmer to have an eye for _____ and _____.

2. The real meaning of style is that you have to have a good sense for discriminating between "good" and "bad" approaches to _____.
3. Adding little features can be very hard because the _____ might not be designed to support that kind of information.
4. Moving one button can be quite a hassle because you may have to go back and change where every single button is located both in the routine to _____ and in the routine to _____.
5. Changing one button wouldn't mean change others, if the button positions are _____.
6. The actual writing of code will be easier if you are willing to put in the _____ before designing your program.
7. In your programming career, you will certainly make small mistakes that cost you hours of debugging only to realize that you were _____ so the compiler thought it was another variable.
8. Full-time programmers must be patient because they will spend a great deal of time working on _____ for other programmers and possibly even _____ in someone else's code.
9. Thinking in a logical, precise and rigorous way means that you have to be willing and able to _____ in a process and understand exactly what goes into it.
10. A successful programmer must be capable of _____ the right way and be or become a good problem solver.

Vocabulary Exercise

IV. Fill in the blanks with the given words. You may not use any of the words more than once. Change the form if necessary.

A.

mediocre	rigorous	hypothesis	hassle	logical
bleak	eliminate	mitigate	persistent	govern
discriminate	reframe	caught	brittle	myth
disciplined	take	exhausting	trait	variable

1. How well a person recovers from stroke is highly _____ and highly individual.
2. In his 200 interviews, Belk found that the most obsessive collectors had one overwhelming _____ in common: loneliness.
3. It was so clear that everything was carefully planned; we _____ the possibility that it could have been an accident.