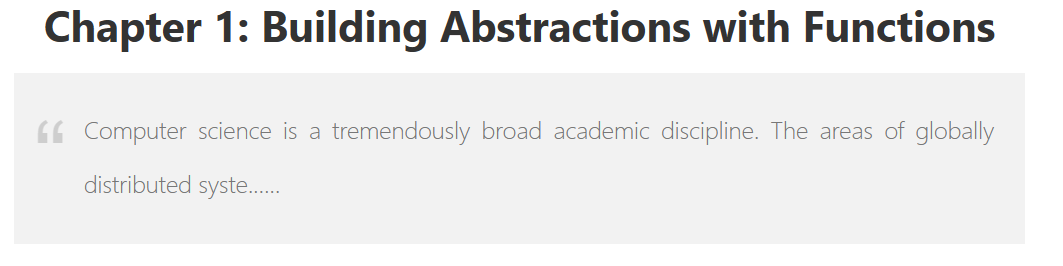
**Chapter 1: Building Abstractions with Functions :**



**Chapter 1: Building Abstractions with Functions**

Computer science is a tremendously broad academic discipline. The areas of globally distributed systems, artificial intelligence, robotics, graphics, security, scientific computing, computer architecture, and dozens of emerging sub-fields each expand with new techniques and discoveries every year. The rapid progress of computer science has left few aspects of human life unaffected. Commerce, communication, science, art, leisure, and politics have all been reinvented as computational domains.

计算机科学是一门极其广泛的学科。全球分布系统、人工智能、机器人技术、图形学、安全性、科学计算、计算机体系结构等领域以及数十个新兴子领域每年都在不断发展新的技术和发现。计算机科学的迅猛进展影响了人类生活的几乎所有方面。商业、通信、科学、艺术、休闲和政治都被重新构想为计算领域。

The tremendous productivity of computer science is only possible because it is built upon an elegant and powerful set of fundamental ideas. All computing begins with representing information, specifying logic to process it, and designing abstractions that manage the complexity of that logic. Mastering these fundamentals will require us to understand precisely how computers interpret computer programs and carry out computational processes.

计算机科学之所以能够取得巨大的生产力，是因为它建立在一组优雅而强大的基本思想之上。所有的计算都始于信息的表示，规定用于处理它的逻辑，并设计用于管理该逻辑复杂性的抽象。掌握这些基本原理需要我们精确理解计算机如何解释计算机程序并执行计算过程。

<https://www.bilibili.com/video/BV1v64y1Q78o?t=822.2>

“这门课很重要的一点是控制程序复杂性(managing their compexity)，我最近才知道一辆汽车有1亿行代码，在我年轻的时候，汽车没有代码，所以这是一个改变，自然，任何一个人都不可能完全掌握一个如此庞大的系统，所以这么多年来我们开发了很多技术来控制程序的复杂性，最大的（技术）是叫做**抽象abstraction，模块化modularization**之类的东西，它们会是本课程的重要部分。”

These fundamental ideas have long been taught at Berkeley using the classic textbook ***Structure and Interpretation of Computer Programs* (**[**SICP**](http://mitpress.mit.edu/sicp)**)** by Harold Abelson and Gerald Jay Sussman with Julie Sussman. These lecture notes borrow heavily from that textbook, which the original authors have kindly licensed for adaptation and reuse.

伯克利大学长期以来一直使用哈罗德·阿贝尔森（Harold Abelson）和杰拉尔德·杰伊·萨斯曼（Gerald Jay Sussman）与朱莉·萨斯曼（Julie Sussman）合著的经典教材**《计算机程序的构造和解释》（Structure and Interpretation of Computer Programs，SICP）**来教授这些基本思想。这些讲义笔者充分借鉴了该教材，原著作者慷慨地授权了对其进行改编和再利用。

The embarkment of our intellectual journey requires no revision, nor should we expect that it ever will.

我们的智力之旅的开始不需要修改，也不应该期望它会有所改变。

We are about to study the idea of a***computational process***. Computational processes are abstract beings that inhabit computers. As they evolve, processes manipulate other abstract things called data. The evolution of a process is directed by a pattern of rules called a program. People create programs to direct processes. In effect, we conjure the spirits of the computer with our spells.

我们将要研究计算**过程的概念**。计算过程是抽象实体，存在于计算机中。随着它们的演变，计算过程会操作其他抽象实体，称为数据。进程的演化受到一组规则模式的指导，这些规则称为程序。人们创建程序来指导进程。实际上，我们通过咒语召唤计算机的精灵。

The programs we use to conjure processes are like a sorcerer's spells. They are carefully composed from symbolic expressions in arcane and esoteric *programming languages* that prescribe the tasks we want our processes to perform.

我们用来召唤进程的程序就像巫师的咒语一样。它们是由奥秘的编程语言中的符号表达式精心构成的，这些符号表达式规定了我们希望进程执行的任务。

A computational process, in a correctly working computer, executes programs precisely and accurately. Thus, like the sorcerer's apprentice, novice programmers must learn to understand and to anticipate the consequences of their conjuring.

在正确工作的计算机中，计算过程会精确而准确地执行程序。因此，就像巫师的学徒一样，新手程序员必须学会理解并预测他们召唤的后果。

—Abelson and Sussman, [SICP](http://mitpress.mit.edu/sicp) (1993)

—阿贝尔森和萨斯曼，SICP（1993）

[**1.1.1   Programming in Python**](https://inst.eecs.berkeley.edu/~cs61a/sp12/book/functions.html#id2)

A language isn’t something you learn so much as something you join.

—[Arika Okrent](http://arikaokrent.com/)

In order to define **computational processes**, we need a programming language; preferably one many humans and a great variety of computers can all understand. In this course, we will learn the [Python](http://docs.python.org/py3k/) language.

为了定义**计算过程**，我们需要一种编程语言，最好是许多人和各种计算机都能理解的语言。在这门课程中，我们将学习Python语言。

Python is a widely used programming language that has recruited enthusiasts from many professions: web programmers, game engineers, scientists, academics, and even designers of new programming languages. When you learn Python, you join a million-person-strong community of developers. Developer communities are tremendously important institutions: members help each other solve problems, share their code and experiences, and collectively develop software and tools. Dedicated members often achieve celebrity and widespread esteem for their contributions. Perhaps someday you will be named among these elite Pythonistas.

Python是一种广泛使用的编程语言，吸引了来自许多不同领域的爱好者：网页程序员、游戏工程师、科学家、学者，甚至是新编程语言的设计者。当你学习Python时，你加入了一个拥有数百万开发者的社区。开发者社区是极其重要的机构：成员们互相帮助解决问题，分享他们的代码和经验，共同开发软件和工具。一些热心的成员通常因为他们的贡献而在社区中备受尊敬和赞誉。也许有一天你会被列为这些精英Pythonista之一。

The Python language itself is the product of a [large volunteer community](http://www.python.org/psf/members/) that prides itself on the [diversity](http://python.org/community/diversity/) of its contributors. The language was conceived and first implemented by [Guido van Rossum](http://en.wikipedia.org/wiki/Guido_van_Rossum) in the late 1980's. The first chapter of his [Python 3 Tutorial](http://docs.python.org/py3k/tutorial/appetite.html) explains why Python is so popular, among the many languages available today.

Python语言本身是一个由大量志愿者组成的社区的产物，他们以贡献者的多样性自豪。这门语言最初由Guido van Rossum于1980年代末构思和首次实现。他的Python 3教程的第一章解释了为什么Python在当今众多可用语言中如此受欢迎。

Python excels as an instructional language because, throughout its history, Python's developers have emphasized the human interpretability of Python code, reinforced by the [Zen of Python](http://www.python.org/dev/peps/pep-0020/) guiding principles of beauty, simplicity, and readability. Python is particularly appropriate for this course because its broad set of features support a variety of different programming styles, which we will explore. While there is no single way to program in Python, there are a set of conventions shared across the developer community that facilitate the process of reading, understanding, and extending existing programs. Hence, Python's combination of great flexibility and accessibility allows students to explore many programming paradigms, and then apply their newly acquired knowledge to thousands of [ongoing projects](http://pypi.python.org/pypi).

Python作为一种教学语言表现出色，因为在其历史上，Python的开发者一直强调Python代码的可解释性，这得到了Python之禅（Zen of Python）的原则的支持，这些原则包括美观、简单和可读性。Python特别适合这门课程，因为它广泛的功能集支持多种不同的编程风格，我们将会探讨这些风格。虽然在Python中没有单一的编程方式，但在开发者社区中有一组共享的约定，有助于阅读、理解和扩展现有的程序。因此，Python既具有很大的灵活性又易于访问，使学生能够探索多种编程范式，然后将他们新获得的知识应用于成千上万个正在进行中的项目。

These notes maintain the spirit of [SICP](http://mitpress.mit.edu/sicp) by introducing the features of Python in lock step with techniques for abstraction design and a rigorous model of computation. In addition, these notes provide a practical introduction to Python programming, including some advanced language features and illustrative examples. Learning Python will come naturally as you progress through the course.

这些讲义保持了SICP的精神，通过与抽象设计技术和严格的计算模型相结合的方式，逐步介绍了Python的特性。此外，这些讲义还提供了对Python编程的实际介绍，包括一些高级语言特性和示例。随着课程的进行，学习Python会自然而然地发生。

However, Python is a rich language with many features and uses, and we consciously introduce them slowly as we layer on fundamental computer science concepts. For experienced students who want to inhale all of the details of the language quickly, we recommend reading Mark Pilgrim's book [Dive Into Python 3](http://diveintopython3.ep.io/), which is freely available online. The topics in that book differ substantially from the topics of this course, but the book contains very valuable practical information on using the Python language. Be forewarned: unlike these notes, Dive Into Python 3 assumes substantial programming experience.

然而，Python是一门功能丰富的语言，具有许多特性和用途，我们有意地在逐步引入基本计算机科学概念的同时慢慢介绍这些特性。对于有经验的学生，他们想要快速掌握语言的所有细节，我们建议阅读Mark Pilgrim的免费在线书籍《Dive Into Python 3》。该书的主题与本课程的主题有很大的不同，但该书包含了关于使用Python语言的非常有价值的实际信息。请注意：与这些讲义不同，《Dive Into Python 3》假定读者具有相当的编程经验。

The best way to get started programming in Python is to **interact with the interpreter directly**. This section describes how to install Python 3, initiate an interactive session with the interpreter, and start programming.

开始使用Python编程的最佳方式是**直接与解释器**互动。本节描述了如何安装Python 3，启动与解释器的交互会话，并开始编程。

[**1.1.2   Installing Python 3**](https://inst.eecs.berkeley.edu/~cs61a/sp12/book/functions.html#id3)

As with all great software, Python has many versions. This course will use the most recent stable version of Python 3 (currently Python 3.2). Many computers have older versions of Python installed already, but those will not suffice for this course. You should be able to use any computer for this course, but expect to install Python 3. Don't worry, Python is free.

Dive Into Python 3 has detailed [installation instructions](http://diveintopython3.ep.io/installing-python.html) for all major platforms. These instructions mention Python 3.1 several times, but you're better off with Python 3.2 (although the differences are insignificant for this course). All instructional machines in the EECS department have Python 3.2 already installed.

[**1.1.3   Interactive Sessions**](https://inst.eecs.berkeley.edu/~cs61a/sp12/book/functions.html#id4)

In an interactive Python session, you type some Python *code* after the *prompt*, >>>. The Python *interpreter* reads and evaluates what you type, carrying out your various commands.

在交互式的Python会话中，您在提示符 ">>>" 后面键入一些Python代码。Python解释器会读取并评估您键入的内容，执行各种命令。

There are several ways to start an interactive session, and they differ in their properties. Try them all to find out what you prefer. They all use exactly the same interpreter behind the scenes.

有几种启动交互式会话的方法，它们在属性上有所不同。尝试它们以找出您喜欢的方法。它们背后都使用完全相同的解释器。

* The simplest and most common way is to run the Python 3 application. Type python3 at a terminal prompt (Mac/Unix/Linux) or open the Python 3 application in Windows.

最简单和最常见的方法是运行Python 3应用程序。在终端提示符（Mac/Unix/Linux）中键入python3，或者在Windows中打开Python 3应用程序。

* A more user-friendly application for those learning the language is called Idle 3 (idle3). Idle colorizes your code (called syntax highlighting), pops up usage hints, and marks the source of some errors. Idle is always bundled with Python, so you have already installed it.

对于正在学习Python语言的人来说，更友好的应用程序称为Idle 3（idle3）。Idle会为您的代码着色（称为语法高亮显示），弹出使用提示，并标记某些错误的来源。Idle始终与Python捆绑在一起，所以您已经安装了它。

* The Emacs editor can run an interactive session inside one of its buffers. While slightly more challenging to learn, Emacs is a powerful and versatile editor for any programming language. Read the 61A Emacs Tutorial to get started. Many programmers who invest the time to learn Emacs never switch editors again.

Emacs编辑器可以在其缓冲区之一内运行交互式会话。虽然学习成本稍高，但Emacs是任何编程语言的强大和多功能编辑器。**阅读61A Emacs教程**以开始学习。许多投入时间学习Emacs的程序员再也不会切换编辑器了。

In any case, if you see the Python prompt, >>>, then you have successfully started an interactive session. These notes depict example interactions using the prompt, followed by some input.

无论如何，如果您看到Python提示符 ">>>"，那么您已成功启动了交互式会话。这些说明展示了使用提示符的示例交互，然后是一些输入。

>>> 2 + 2

Controls: Each session keeps a history of what you have typed. To access that history, press <Control>-P (previous) and <Control>-N (next). <Control>-D exits a session, which discards this history.

控制：每个会话都会保存您键入的历史记录。要访问历史记录，请按<Control>-P（上一个）和<Control>-N（下一个）。<Control>-D退出会话，这会删除这个历史记录。

在Python的交互式会话中，您可以轻松地测试代码片段，执行数学运算，探索语言的功能，并查看结果。这是学习和调试Python代码的有用工具。

[**1.1.4   First Example**](https://inst.eecs.berkeley.edu/~cs61a/sp12/book/functions.html#id5)

And, as imagination bodies forth

The forms of things to unknown, and the poet's pen

Turns them to shapes, and gives to airy nothing

A local habitation and a name.

—William Shakespeare, A Midsummer-Night's Dream

正如想象力赋予未知事物形态，

而诗人之笔将其塑造成形状，

为虚无赋予了实在的住所和名字。

—威廉·莎士比亚，《仲夏夜之梦》

To give Python the introduction it deserves, we will begin with an example that uses several language features. In the next section, we will have to start from scratch and build up the language piece by piece. Think of this section as a sneak preview of powerful features to come.

为了给Python一个应有的介绍，我们将从一个使用多种语言特性的示例开始。在下一节中，我们将不得不从零开始，逐步构建语言的各个部分。将这一节视为即将介绍的强大功能的提前预览。

Python has built-in support for a wide range of common programming activities, like manipulating text, displaying graphics, and communicating over the Internet. The import statement

>>> from urllib.request import urlopen

loads functionality for accessing data on the Internet. In particular, it makes available a function called urlopen, which can access the content at a uniform resource locator (URL), which is a location of something on the Internet.

Python具有内置支持广泛的常见编程活动的功能，如操作文本、显示图形和在互联网上通信。import语句

>>> from urllib.request import urlopen

加载了用于访问互联网上数据的功能。特别地，它提供了一个名为urlopen的函数，该函数可以访问统一资源定位器（URL）的内容，URL是互联网上某个事物的位置。

**Statements & Expressions.** 语句和表达式。

Python code consists of statements and expressions. Broadly, computer programs consist of instructions to **either**

Python代码由语句和表达式组成。广义而言，计算机程序包含了**要么**

1. Compute some value  计算某个值
2. Carry out some action  执行某些操作

Statements typically describe actions. When the Python interpreter executes a statement, it carries out the corresponding action. On the other hand, expressions typically describe computations that **yield values**产生值. When Python evaluates an expression, it computes its value.

语句通常描述了操作。当Python解释器执行语句时，它执行相应的操作。另一方面，表达式通常描述了产生值的计算。当Python评估表达式时，它计算出其值。

This chapter introduces several types of statements and expressions.

本章介绍了几种类型的语句和表达式。

The assignment statement 赋值语句

>>> shakespeare = urlopen('http://inst.eecs.berkeley.edu/~cs61a/fa11/shakespeare.txt')

**associates** the name shakespeare with the value of the expression that follows. That expression applies the urlopen function to a URL that contains the complet e text of William Shakespeare's 37 plays, all in a single text document.

将名称shakespeare 与随后的表达式的值**关联**起来。该表达式将urlopen 函数应用于包含威廉·莎士比亚37部剧作全部内容的URL，这些剧作都包含在一个文本文件中。

**Functions**. 函数。

Functions encapsulate封装 logic that manipulates data. A web address is a piece of data, and the text of Shakespeare's plays is another. The process by which the former **leads to** the latter may be complex, but we can apply that process using only a simple expression because that complexity **is tucked away** within a function.

函数封装了操作数据的逻辑。Web地址是一种数据，莎士比亚剧作的文本是另一种数据。前者导致后者的过程可能很复杂，但我们可以只使用简单的表达式来应用该过程，因为该复杂性被隐藏在函数内部。

Functions are the primary topic of this chapter.

函数是本章的主要话题。

Another assignment statement       另一个赋值语句

>>> words = set(shakespeare.read().decode().split())

associates the name words to the set of all **unique** words that appear in Shakespeare's plays, all 33,721 of them. The chain of commands to read, decode, and split, each operate on an intermediate computational entity: data is read from the opened URL, that data is decoded into text, and that text is split into words. All of those words are placed in a set.

将名称words 与出现在莎士比亚剧作中的所有**唯一**单词的集合相关联，总共有33,721个单词。在由读取read、解码decode和拆分split组成的一系列命令（chain of commands ，命令链）中，每个命令都在中间计算实体上操作：从打开的URL读取数据，将数据解码为文本，将文本拆分为单词。所有这些单词都放在一个集合中。

**Objects**. 对象。

A set is a type of object, one that supports set operations like computing intersections and **testing membership**. An object seamlessly bundles together data and the logic that manipulates that data, **in a way that hides the complexity of both**. Objects are the primary topic of Chapter 2.

**对象。**集合是一种对象类型，它支持集合操作，如计算交集和**测试成员资格**。对象以一种无缝的方式捆绑数据和操纵该数据的逻辑，以一种既**隐藏了数据复杂性又隐藏了逻辑复杂性的方式**。对象是第2章的主要话题。

The expression   表达式

>>> {w for w in words if len(w) >= 5 and w[::-1] in words}

{'madam', 'stink', 'leets', 'rever', 'drawer', 'stops', 'sessa',

'repaid', 'speed', 'redder', 'devil', 'minim', 'spots', 'asses',

'refer', 'lived', 'keels', 'diaper', 'sleek', 'steel', 'leper',

'level', 'deeps', 'repel', 'reward', 'knits'}

is a compound expression that evaluates to the set of Shakespearian words that appear both forward and in reverse. The cryptic notation w[::-1] enumerates each letter in a word, but the -1 says to step backwards (:: here means that the positions of the first and last characters to enumerate are defaulted.) When you enter an expression in an interactive session, Python prints its value on the following line, as shown.

这串代码是一个复合表达式，它计算出既以正向又以反向方式出现的莎士比亚词汇集合。这里的w[::-1] 的符号表示对单词中的每个字母进行枚举，但是 -1 表示向后遍历（:: 在这里表示将要枚举的第一个和最后一个字符的位置默认设置为默认值）。当您在交互式会话中输入一个表达式时，Python会在下一行打印出其值，如所示。

**Interpreters**. 解释器。

Evaluating compound expressions requires a precise procedure that interprets code in a predictable way. A program that implements such a procedure, evaluating compound expressions and statements, is called an interpreter. The design and implementation of interpreters is the primary topic of Chapter 3.

评估复合表达式需要一个精确的过程，以可预测的方式解释代码。实现这种过程的程序，用于评估复合表达式和语句，被称为解释器。解释器的设计和实现是第3章的主要话题。

When compared with other computer programs, interpreters for programming languages are unique in their generality. Python was not designed with Shakespeare or palindromes in mind. However, its great **flexibility** allowed us to process a **large** amount of text with only a **few** lines of code.

与其他计算机程序相比，编程语言的解释器在其通用性上具有独特性。Python并不是设计用于处理莎士比亚或回文（palindromes）等内容。然而，它的极大**灵活性**使我们能够仅需几行代码处理大量文本。

In the end, we will find that all of these core concepts are closely related: functions are objects, objects are functions, and interpreters are **instances** of both. However, developing a clear understanding of each of these concepts and their role in organizing code is critical to mastering the **art of programming**.

最终，我们将发现所有这些核心概念都密切相关：函数是对象，对象是函数，解释器是两者的**实例**。然而，对每个概念及其在代码组织中的作用有清晰的理解对掌握编程艺术至关重要。

[**1.1.5   Practical Guidance: Errors**](https://inst.eecs.berkeley.edu/~cs61a/sp12/book/functions.html#id6)

Python is waiting for your command. You are encouraged to experiment with the language, even though you may not yet know its full vocabulary and structure. However, be prepared for errors. While computers are tremendously fast and flexible, they are also extremely rigid. The nature of computers is described in [Stanford's introductory course](http://www.stanford.edu/class/cs101/code-introduction.html) as

Python正在等待您的命令。鼓励您尝试使用这门语言，即使您可能还不完全了解它的全部词汇和结构。但是，请准备好迎接错误。尽管计算机非常快速和灵活，但它们也非常严格。斯坦福大学的入门课程中描述计算机的性质如下：

The fundamental equation of computers is: computer = powerful + stupid

计算机的基本方程是：计算机 = 强大 + 愚蠢

Computers are very powerful, looking at volumes of data very quickly. Computers can perform billions of operations per second, where each operation is pretty simple.

计算机非常强大，可以快速处理大量数据。计算机每秒可以执行数十亿次操作，其中每个操作都相当简单。

Computers are also shockingly stupid and fragile. The operations that they can do are extremely rigid, simple, and mechanical. The computer lacks anything like real insight .. it's nothing like the HAL 9000 from the movies. If nothing else, you should not be intimidated by the computer as if it's some sort of brain. It's very mechanical underneath it all.

计算机也非常愚蠢和脆弱。它们能够执行的操作非常刚性、简单和机械化。计算机缺乏真正的洞察力...它与电影中的HAL 9000等完全不同。最起码，您不应该被计算机吓倒，好像它是某种大脑。在所有这一切的底层，计算机其实非常机械化。

Programming is about a person using their real insight to build something useful, constructed out of these teeny, simple little operations that the computer can do.

—Francisco Cai and Nick Parlante, Stanford CS101

在所有这些之外，编程是一个人利用他们的真正洞察力，构建出有用的东西，由计算机可以执行的这些微小而简单的操作组成。

—Francisco Cai和Nick Parlante，斯坦福大学CS101

The rigidity of computers will immediately become apparent as you experiment with the **Python interpreter**: even the smallest spelling and formatting changes will cause unexpected outputs and errors.

计算机的严格性将立即在您尝试使用**Python解释器**时显现出来：即使是最小的拼写和格式更改也会导致意外的输出和错误。

Learning to interpret errors and diagnose the cause of unexpected errors is called ***debugging***. Some guiding principles of debugging are:

学会解释错误并诊断意外错误的原因被称为***调试*** 。一些调试的指导原则包括：

1. **Test incrementally**: Every well-written program is composed of small, modular components that can be tested individually. Test everything you write as soon as possible to catch errors early and gain confidence in your components.

逐步测试：每个编写良好的程序都由可以单独测试的小型模块组成。尽早测试您编写的所有内容，以尽早捕获错误并增强对组件的信心。

1. **Isolate errors**: An error in the output of a compound program, expression, or statement can typically be attributed to a particular modular component. When trying to diagnose a problem, trace the error to the smallest fragment of code you can before trying to correct it.

隔离错误：复合程序、表达式或语句的输出中的错误通常可以归因于特定的模块组件。在尝试诊断问题时，将错误追踪到尽可能小的代码片段，然后尝试更正它。

1. **Check your assumptions**: Interpreters do carry out your instructions to the letter --- no more and no less. Their output is unexpected when the behavior of some code does not match what the programmer believes (or assumes) that behavior to be. Know your assumptions, then focus your debugging effort on verifying that your assumptions actually hold.

检查您的假设：解释器会严格按照您的指示执行 —— 不多也不少。当某些代码的行为不符合程序员认为（或假设）的行为时，其输出就会出现意外。了解您的假设，然后将调试工作重点放在验证您的假设是否确实成立上。

1. **Consult others**: You are not alone! If you don't understand an error message, ask a friend, instructor, or search engine. If you have isolated an error, but can't figure out how to correct it, ask someone else to take a look. A lot of valuable programming knowledge is shared in the context of team problem solving.

咨询他人：您并不孤单！如果您不理解错误消息，可以向朋友、教师或搜索引擎提问。如果您已经隔离了错误，但不知道如何纠正它，可以请别人过来看看。在团队问题解决的背景下，有许多有价值的编程知识是共享的。

Incremental testing, modular design, precise assumptions, and teamwork are themes that persist throughout this course. Hopefully, they will also persist throughout your computer science career.

逐步测试、模块化设计、精确的假设和团队合作是贯穿本课程的主题。希望它们也会贯穿于您的计算机科学职业生涯中。

A programming language is more than just a means for instructing a computer to perform tasks. **The language also serves as a framework within which we organize our ideas about processes.** Programs serve to communicate those ideas among the members of a programming community. Thus, programs must be written for people to read, and only incidentally for machines to execute.

编程语言不仅仅是一种指导计算机执行任务的手段。**语言还充当我们组织关于进程的思想的框架**。程序用于在编程社区的成员之间传达这些思想。因此，程序必须是为人们阅读而编写的，只是偶然地为机器执行。

这里的 "processes" 指的是计算机程序中执行的一系列操作或任务。在编程和计算机科学中，"processes" 通常指的是程序运行时所涉及的一系列步骤、操作或任务，这些步骤按照特定的顺序或逻辑执行，以完成特定的计算任务或处理数据。这些 "processes" 可能包括数据处理、算法执行、用户界面交互等等。

这句话的意思是编程语言不仅仅是一种用于指导计算机执行任务的工具，还提供了一种框架，用于组织我们关于计算过程的思想。编程语言帮助我们以一种结构化和可管理的方式描述和实现各种计算过程。

When we describe a language, we should pay particular attention to the means that the language provides for combining simple ideas to form more complex ideas. Every powerful language has three mechanisms for accomplishing this:

当我们描述一种语言时，我们应特别关注语言提供的方式，通过这些方式，可以将简单的思想组合成更复杂的思想。**每种强大的语言**都有三种机制来实现这一点：

* **primitive expressions and statements**, which represent the simplest **building blocks** that the language provides,
* 原始表达式和语句，代表语言提供的最简单的**构建块**，
* **means of combination**, by which compound elements are built from simpler ones, and
* 组合方式，通过该方式从较简单的元素构建复合元素，
* **means of abstraction**, by which compound elements can be named and manipulated as units.
* 抽象方式，通过该方式可以将复合元素命名并作为单位进行操作

In programming, we deal with two kinds of elements: functions and data. (Soon we will discover that they are really not so distinct.) Informally, data is stuff that we want to manipulate, and functions describe the rules for manipulating the data. Thus, any powerful programming language should be able to describe primitive data and primitive functions and should have methods for combining and abstracting both functions and data.

在编程中，我们处理两种类型的元素：函数和数据。（很快我们会发现它们实际上并不那么不同。）非正式地说，数据是我们想要操作的东西，而函数描述了操作数据的规则。因此，任何强大的编程语言都应该能够描述原始数据和原始函数，并且应该有一种方法来组合和抽象函数和数据。

**#SuperMemo Reference:  
#Title: Chapter 1: Building Abstractions with Functions  
#Date: Imported on 2023-09-10 13:38:49  
#Source: Web import  
#Link:** [**https://inst.eecs.berkeley.edu/~cs61a/sp12/book/functions.html**](https://inst.eecs.berkeley.edu/~cs61a/sp12/book/functions.html) **#Article: #2365 (deleted)  
#Parent:** [**4644**](file:///E:\Git-SuperMemobackup\SuperMemoBackup\sm18-lazy-package-1.3.1\sm18\systems\ALL%20IN%20ONE\temp\SuperMemoElementNo=(4644))**: CS61A: Online Textbook This book is derived from the classic textbook Structure and Interpretation of Computer Programs by Abelson, Sussman, and Sussman. John Denero .  
#Concept group: Root (**[**4644**](file:///E:\Git-SuperMemobackup\SuperMemoBackup\sm18-lazy-package-1.3.1\sm18\systems\ALL%20IN%20ONE\temp\SuperMemoElementNo=(4644))**: CS61A: Online Textbook This book is derived from the classic textbook Structure and Interpretation of Computer Programs by Abelson, Sussman, and Sussman. John Denero .)**