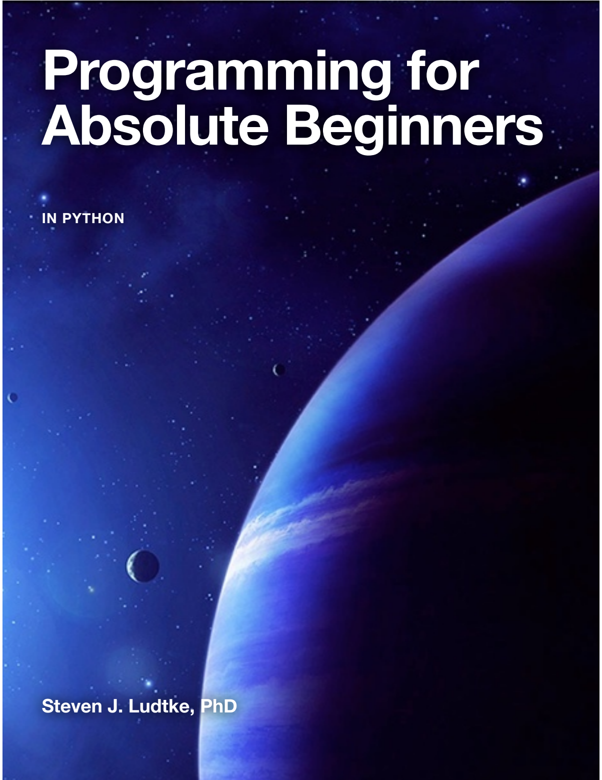
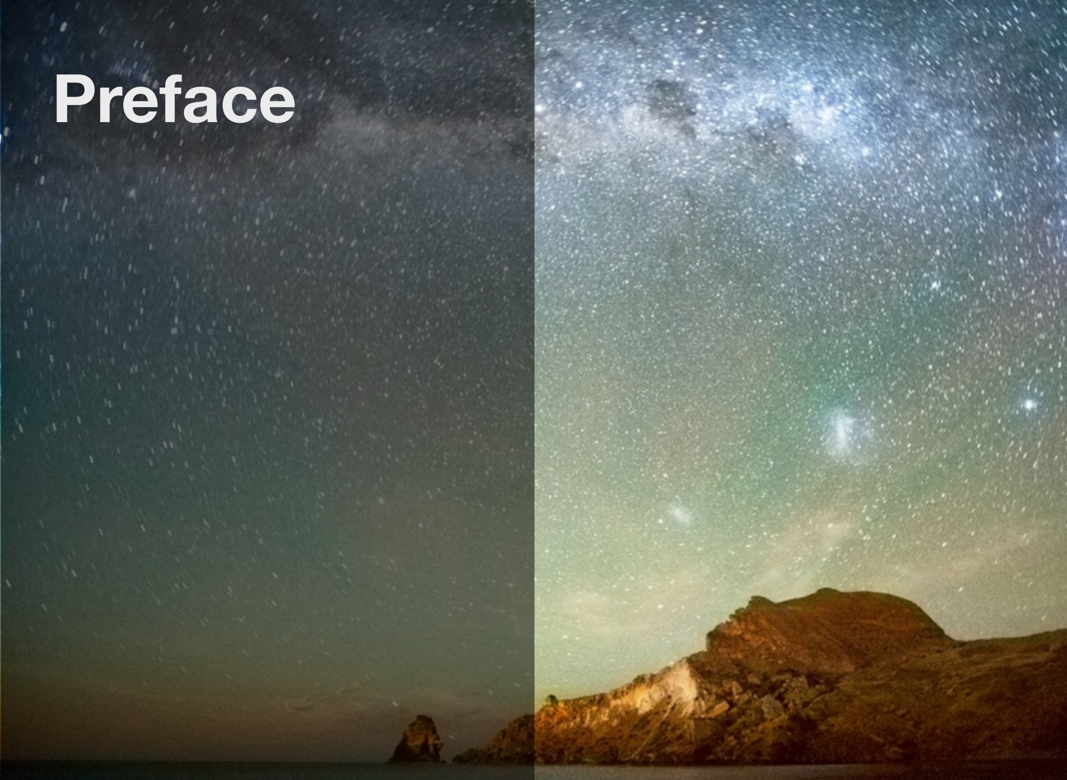


Programming for Absolute Beginners

IN PYTHON

Steven J. Ludtke, PhD

Preface



Preface

I first learned to program when I was about 10 years old, when my father purchased our first computer, a TRS-80 Model I. Most of you have probably never heard of this computer, as the year was 1978, and this was one of the first ‘personal computers’ to come to market. There were no home video games yet, and such devices were quite rare. In those days, there were no hard drives, and even floppy drives weren’t available yet. The sole mechanism for storing programs was on an audio cassette tape. The computer had the ‘BASIC’ programming language built in to it, but otherwise came with no software at all. If you wanted to play a game, you had to type in a program from a magazine first. Needless to say, such games were primitive even by the standards of what you might find on a cell-phone ten years ago. Nonetheless, it was cutting edge at the time.

My father made a deal with me. I could play games on the computer, but only if I went through the ‘learn to program’ book that came with the computer. This book was really quite amazing even by modern standards. It had little cartoons, and took you step by step through the art of programming in BASIC. I still remember bits of it, even 30 years later, and once I started learning how to get these fascinating machines to do something, I was hooked. After all, you really couldn’t do much interesting on the machines without knowing how to program.

Of course, The world has changed. Nowadays, you can buy or download a program for just about any task you can imagine. Some of these programs even include little programming languages of their own.

So, why learn to program at all ? Seriously, why ?

Chapter 1

Getting Ready to Program

Python is, itself, a program you run on your computer, which interprets the programs you enter. This chapter takes you through the process of setting your computer up to use Python, and some initial examples to show you what Python can do.

Section 1

How This Book Works

SUMMARY

1. ***Text to type***
2. *Text you see on the screen*
3. <key> to press
4. Example code at
5. Chapter ? contains a review of concepts

Conventions

I tried to write this book so it would be understandable to someone learning programing for the very first time. If you already know a little bit of programming from somewhere, you could also use this book to learn Python, though you may find some of it a little simplistic. To add a little spice, you will find occasional boxes that look like this: box, with notes for more advanced readers. If you're a beginner, you can ignore these.

There are a few basic conventions we use in this book that it's important to be familiar with, so you find the process fun, not frustrating. Most people will find these things pretty obvious, but let's just make sure everyone's on the same page. First, if you see ***text that looks like this***, you are supposed to (or can if you like) type exactly what you see into the computer. Generally you should press <enter> at the end of each line. When you see something between <> characters, this represents a key you are supposed to press, such as <p>, <space> or <enter>. Please note also that <enter> is the same as <return>. Different keyboards give it different names. Sequences like <ctrl-c> mean you should hold down the 'ctrl' key and press c. Text *like this* is text you should expect to see displayed on the screen.

Section 2

Installing Python

SUMMARY

1. **Python installation is platform-dependent (Linux, Mac, Windows).**
2. **This book will use Python 2.7.x. Most of what is covered will also apply to other versions.**
3. **We also install some useful Python libraries which aren't part of the standard distribution.**
4. **Wherever text appears in *`this font`*, this is an indication of something you are expected to type into your computer.**

The Good

Python is, quite simply, a fun language to use. Whereas many programming languages force you to do any specific task one specific way, and make you carefully define every aspect of your program before you can actually do anything, Python is very relaxed, and free-form. For any given task, it is generally possible to come up with a half-dozen different ways to accomplish it in Python. While it permits you to be very rigid in your software design, it also gives you the freedom to simply play around (which is what we will spend most of this book doing).

Python is widespread enough that it is included as a standard part of most (but not all) modern operating systems. The specific version of Python you will have will vary with how old your OS is. The exercises in this book will focus on Python 2.7.x. The vast majority of what we cover will also be valid for earlier versions of Python. Python 3.x has also been available now for some time, however adoption has been slow because it isn't 100% compatible with earlier versions. We will try to mention any places where there is a critical difference between Python 3 and Python 2.

The Bad

Python is what is known as an interpreted programming language. When you write a program in a language like C++ or Fortran, your program is first passed through

Section 3

Taking Python Out for a Spin

SUMMARY

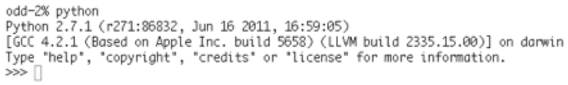
1. You can start Python by typing ***python*** at the command-prompt.
2. Python can be used to do basic math like a calculator, for example ***2*5+10***. If you need scientific functions, like sqrt() or cos(), first you have to type: ***from math import ****
3. A string can be created by surrounding text with double quotes, such as : ***"a test"***. You can also perform addition and multiplication with strings.
4. Python has built-in Turtle graphics, which can be used to do simple drawing operations. This emulates a real Turtle robot drawing with a pen.

Starting Python

There are two fundamentally different ways you can use an **interpreted language** like Python. First, you can use a text editor to create a file containing your program, then you can run the program just like you do any other application on your computer. Alternatively, you can run Python in **interactive mode**, and just type commands into it one after another. It will immediately respond to each command. We will make use of both methods in this book. However, we will begin with the interactive mode, and use this for many of the simple exercises in the book.

On any of the three computer platforms we cover, Python can be run by opening a command prompt, and typing ***python***. While you can start it using an Icon on most platforms as well, there are some reasons not to do it that way just yet.

So, go ahead and give it a try. Once you enter ***python***, you should receive a prompt, looking something like:



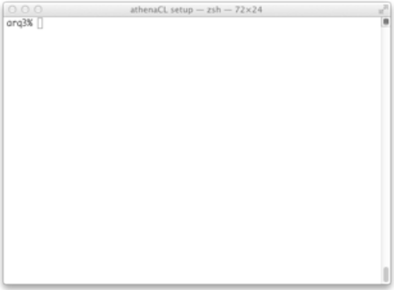
```
odd-2% python
Python 2.7.1 (r271:86832, Jun 16 2011, 16:59:05)
[GCC 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2335.15.00)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Problems

While you haven't really been 'taught' anything yet, if you're clever, you may be able to figure out the examples you've seen enough to try your hand at a few simple problems. Of course, the answers are provided as well.

Problem 1 - Print the integers from 0 to 10 and the square root of each.

Solution 1.1



```
arq3% 
```

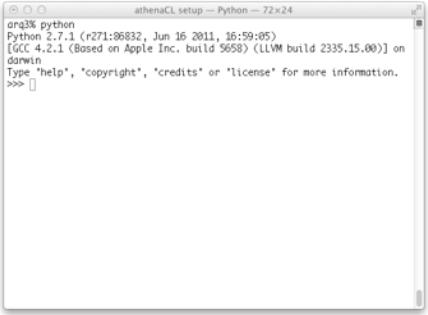
Scroll through the images to see the solution.

Problem 2 - Modify the turtle examples and see if you can draw:

a) A hexagon

b) The spirograph example is loosely based on triangles, modify it so its based on squares instead.

Solution 1.2



```
arq3% python
Python 2.7.1 (r271:86832, Jun 16 2011, 16:59:05)
[GCC 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2335.15.00)] on
darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Again, start by running ***`python`***.

Chapter 2

Turtlerific

Turtles have the most potential for doing something interesting quickly, so we'll take a couple of turtle examples apart to see what we can learn from them.

Section 1

Spirograph Example

SUMMARY

1. **Python has many built in modules, including math and the turtle graphics module we have already used. To use a module you must either *`import module`* or *`from module import *`*.**
2. **Python has a built-in help function, which can be used to get documentation for modules or functions, such as *`help(math)`*.**
3. **There are over 20 different commands you can give the turtle. The most useful of these are summarized.**
4. **Lists can be created using square brackets and commas, such as: *`[1,2,3,4]`*.**
5. **The *`for`* loop allows us to repeat an operation for each element in a list.**

Let's start with our spirograph example from the last chapter:

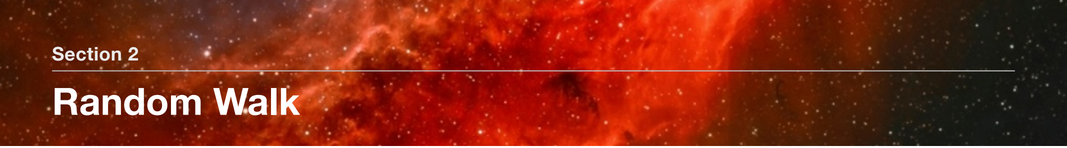
```
from turtle import *
s=Screen()
reset()
goto(-125,-125)
clear()
for i in range(61):
    forward(250)
    left(118)
```

This example shouldn't be too difficult to figure out. Let's start with the turtle graphics functions: *Screen(), reset(), goto(), clear(), forward()* and *left()*. These functions wouldn't be available except for the first line:

```
from turtle import *
```

Import

Python comes with a wide range of standard libraries to do all sorts of useful and interesting things. We've seen two of these libraries in the examples in the first chapter: math and turtle. While these libraries are distributed with the Python



Section 2

Random Walk

LOREM IPSUM

1. **The random module provides functions for making random numbers of different sorts.**
2. **Less Random Walks**
3. **Traveling Circle**
4. **Making Decisions**

Totally Random Walks

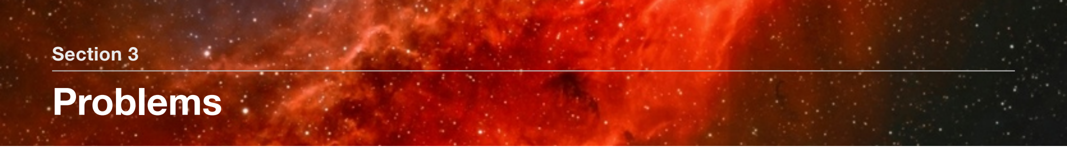
So far, we've introduced two modules: math and turtle. Let's go ahead and add one more to our repertoire. Try this:

```
import random
for i in range(10): print random.randint(1,100)
```

As you'll see, this program will print 10 random numbers between 1 and 100 (possibly including 100). If you run the program again, you'll get a different list of numbers each time. There are a number of other functions available within the random module as well, for example, ***random.uniform(1,100)*** will return a random floating point number between 1 and 100. ***random.gauss(80,10)*** will return a 'Gaussian' (a bell-shaped curve) centered at 80, with a width of 10. That is it will be more likely to return values close to 80. The farther you get from 80, the less likely it is to produce that number, but technically it could return 1000. It's simply very unlikely.

Let's try applying this to turtle graphics:

```
from random import *
from turtle import *
a=Turtle()
speed(0)
```



Section 3

Problems

1)

Chapter 3

Sudoku

In this chapter, we'll learn how to create and solve Sudoku puzzles. Even if you don't know Sudoku or don't like them, this chapter will introduce many useful concepts.

Section 1

Making A New Sudoku

LOREM IPSUM

Sudoku, if you aren't familiar with it, is a popular puzzle game of pattern completion. While the rules are simple, they can be extremely challenging to solve.

The Rules

Before we can start thinking about how to write a program to solve or create Sudoku puzzles, we need to understand the rules. If you're already familiar with the rules, you can skip this section.

Sudoku is a pretty simple game. It's played on a 9 x 9 square grid. each square holds a number between 1 and 9. The trick ? Each row, each column and each 3x3 smaller square can only have one of each of the 9 numbers. An example of a solved Sudoku is shown to the right. When you have a book of Sudoku puzzles, some of the numbers are missing. The goal of the game is to fill in all of the missing numbers. The more that are missing, the harder the puzzle.

How would you go about making a Sudoku from scratch ? One simple approach would be to begin with an empty grid, then start filling in random numbers in squares. Each time you insert a number, you check to see if it's legal. If

1	2	3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
2	3	4	5	6	7	8	9	1
5	6	7	8	9	1	2	3	4
8	9	1	2	3	4	5	6	7
3	4	5	6	7	8	9	1	2
6	7	8	9	1	2	3	4	5
9	1	2	3	4	5	6	7	8

A Solved Sudoku

Section 2

Nesting and Recursion

SUMMARY

1. **A nested loop is a loop inside another loop. It is used to loop over more than one dimension. For example, all of the rows and columns in a table.**
2. **A recursive function is a function that calls itself. Recursion has many uses, and while it is a bit difficult to absorb, in some cases there is virtually no other good way of achieving the same results.**

So far we've gotten by with simple ***for*** loops and lists. Before we move on, we're going to introduce not one, but two core concepts in programming: nested loops and recursive functions.

Nested Loops

Nested loops really aren't a very difficult concept to grasp, but sometimes it can take a little practice to understand when to use them. The ***for*** loops we've already used operate on a single list. Now consider a table (like the 9 x 9 elements in our Sudoku puzzle). While it's absolutely possible to do what we did with the Sudoku puzzle, and 'unwrap' our table into a single list, it can also make our programs a lot more complicated. Let's take a look at a simple problem: printing a multiplication table. We'll do the same thing two different ways, one using a single ***for*** loop, and one using a nested loop.

The Old Approach

If we want to make a multiplication table with a single ***for*** loop, we need to know the total number of entries in the table, and then figure out which row and column we're in for each item in the loop. We also have to jump through a few hoops to decide when to start a new line. For simplicity, we'll do a 9x9 multiplication table. That means (like the Sudoku) we will have 81 elements in our table. We can use

Section 3

Solving Sudoku

SUMMARY

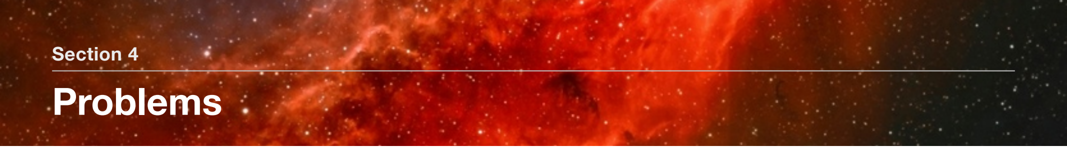
Let's turn the problem around now. Let's say we have a Sudoku with missing numbers. How would we go about filling in the missing values (correctly) ?

Regardless of how we try to fill the numbers in, we will need to define a function that checks to see if a given Sudoku list is a legal solution or not. This is pretty straightforward conceptually. We just need to loop over all of the rows, columns and 3x3 regions, and check if all of them have exactly 1 of each number 1-9. If any fail the test, then we return False. Otherwise we return True.

Sets

To do this efficiently, we need to introduce yet another type of Python object: the ***set***. Like a ***list***, a ***set*** contains other objects, such as numbers or strings. Unlike a ***list***, a ***set*** has no order, and the items in the ***set*** are unique. That is, if you have a set containing 1,3 and 5, then add 3 to the set, the set will still have 1,3 and 5 in it. With a ***list***, if you had ***[1,3,5]***, and appended 3 to the ***list***, you would have ***[1,3,5,3]***. Give this a try:

```
a=[1,2,3,4,3,2,5,7,9,12,3]
b=set(a)
print a,b
```



Section 4

Problems

Lorem ipsum dolor sit amet, ligula suspendisse nulla pretium, rhoncus tempor placerat fermentum, enim integer ad vestibulum volutpat. Nisl rhoncus turpis est, vel elit, congue wisi enim nunc ultricies sit, magna tincidunt. Maecenas aliquam maecenas ligula nostra, accumsan taciti. Sociis mauris in integer, a dolor netus non dui aliquet, sagittis felis sodales, dolor sociis mauris, vel eu est libero cras. Interdum at. Eget habitasse elementum est, ipsum purus pede porttitor class, ut lorem adipiscing, aliquet sed auctor, imperdiet arcu per diam dapibus libero duis. Enim eros in vel, volutpat nec pellentesque leo, temporibus scelerisque nec.

Ac dolor ac adipiscing amet bibendum nullam, massa lacus molestie ut libero nec, diam et, pharetra sodales eget, feugiat ullamcorper id tempor eget id vitae. Mauris pretium eget aliquet, lectus tincidunt. Porttitor mollis imperdiet libero senectus pulvinar. Etiam molestie mauris ligula eget laoreet, vehicula eleifend. Repellat orci eget erat et, sem cum, ultricies sollicitudin amet eleifend dolor nullam erat, malesuada est leo ac.

Varius natoque turpis elementum est. Duis montes, tellus lobortis lacus amet arcu et. In vitae vel, wisi at, id praesent bibendum libero faucibus porta egestas, quisque praesent ipsum fermentum placerat tempor. Curabitur auctor, erat mollis sed fusce, turpis vivamus a dictumst congue magnis. Aliquam amet ullamcorper dignissim molestie, sed mollis. Tortor vitae tortor eros wisi facilisis. Consectetuer arcu ipsum ornare pellentesque vehicula, in vehicula diam, ornare magna erat felis wisi a risus. Justo fermentum id. Malesuada eleifend, tortor molestie, a fusce a vel et. Mauris at suspendisse, neque aliquam faucibus adipiscing, vivamus in.

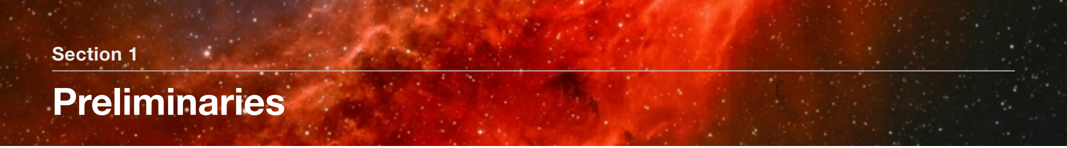
Wisi mattis leo suscipit nec amet, nisl fermentum tempor ac a, augue in eleifend in ipsum venenatis, cras sit id in vestibulum felis in, sed ligula. In sodales suspendisse mauris quam etiam erat, quia tellus convallis eros rhoncus diam orci, porta lectus esse adipiscing posuere et, nisl arcu vitae laoreet. Morbi integer molestie, amet suspendisse morbi, amet, a maecenas mauris neque proin nisl mollis.

Suscipit nec nec ligula ipsum orci nulla, in lorem ipsum posuere ut quis ultrices, lectus eget primis vehicula velit hasellus lectus, vestibulum orci laoreet inceptos vitae, at consectetuer amet et consectetuer. Congue porta scelerisque praesent at, lacus vestibulum et at dignissim cras urna.

Chapter 4

Wordgames

In this chapter, we’ll use strings in Python to provide solutions for some common word-games, or perhaps, even create some new puzzles.



Section 1

Preliminaries

LOREM IPSUM

More Ify Statements

While we could just jump right in and start writing programs again. Let's take a break and introduce a few more concepts first.

Let's start by going back briefly to our discussion of the ***if*** statement. Actually, there's even more to it than we've seen. The full description of ***if*** is:

```
if expression : do something
elif expression : do something else
else : do something completely different
```

This sequence of commands lets you ask a sequence of linked questions. If the first question is *True*, then it executes **do something** and skips the rest. If the first question is *False*, then it asks the second question. If that's *True*, then it executes **do something else**, and skips the *else*. If that one isn't *True* either, it executes **do something completely different**. You can put as many *elif* statements in as you like, but there can only be a single *else* at the end, which happens only when everything else in the list is false.

This sequence is often used to do things like ask the users questions. Let's try using ***raw_input*** with our new extended ***if*** statement:

Section 2

What Words Can You Make ?

LOREM IPSUM

1. **Lorem ipsum dolor sit amet**
2. **Consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.**
3. **Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.**
4. **Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.**

There are a lot of games which involve making words from random letters, some involve dice, some involve tiles. What they all share in common is a set of random letters.

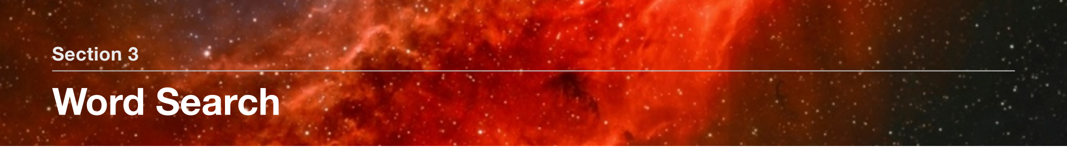
Once again, when settling down to write a program to do something, the first thing we need to do is describe exactly what the program would do, step by step. Think of it as explaining the steps of the process to a six year old. You need to cover every step in detail. The problem we want to solve in this section is to take a list of N (could be 5 or 7 or whatever you like) letters, and rearrange them to produce any possible word.

Break it Down

Now that we have a broad definition of the problem we want to solve we need to go into more detail. We'll start out with a simplified version of the problem. Say, we have 5 letters: A,B,C,D,E. In most games, the rule is that you can use each letter only once when you make words. We'll deal with that eventually, but let's start out without that restriction, just to make things easier. That is, we have an "A", so we can use "A" as many times as we like. Let's summarize the rules of our little "game":

We have 5 letters

Each letter can be used as many times as we like



Section 3

Word Search

LOREM IPSUM

1. **Lorem ipsum dolor sit amet**
2. **Consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.**
3. **Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.**
4. **Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.**

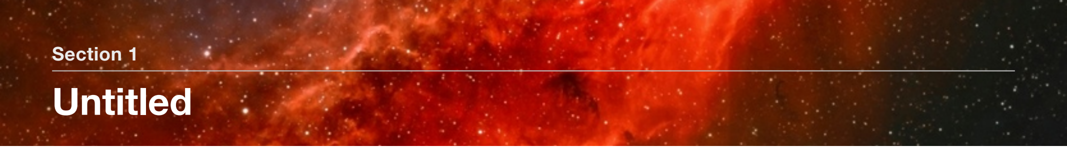
Lorem ipsum dolor sit amet, ligula suspendisse nulla pretium, rhoncus tempor placerat fermentum, enim integer ad vestibulum volutpat. Nisl rhoncus turpis est, vel elit, congue wisi enim nunc ultricies sit, magna tincidunt. Maecenas aliquam maecenas ligula nostra, accumsan taciti. Sociis mauris in integer, a dolor netus non dui aliquet, sagittis felis sodales, dolor sociis mauris, vel eu libero cras. Interdum at. Eget habitasse elementum est, ipsum purus pede porttitor class, ut adipiscing, aliquet sed auctor, imperdiet arcu per diam dapibus libero duis. Enim eros in vel, lorem ispum volutpat nec pellentesque leo, temporibus scelerisque nec. Ac dolor ac adipiscing amet bibendum nullam, massa lacus molestie ut libero nec, diam et, pharetra sodales eget, feugiat ullamcorper id tempor eget id vitae. Mauris pretium eget aliquet, lectus tincidunt. Porttitor mollis imperdiet lorem ipsum libero senectus pulvinar.

Etiam molestie mauris ligula eget laoreet, vehicula eleifend. Repellat orci eget erat et, sem cum, ultricies sollicitudin amet eleifend dolor nullam erat, malesuada est leo ac. Varius natoque turpis elementum est. Massa lacus molestie ut libero nec, diam et, pharetra sodales eget, feugiat ullamcorper id tempor eget id vitae. Mauris pretium eget aliquet, lectus tincidunt. Porttitor mollis imperdiet libero senectus pulvinar. Etiam molestie mauris ligula eget laoreet, vehicula eleifend. Repellat orci eget erat et, sem cum.

Chapter 5

MATH

Lorem ipsum dolor sit amet, ligula suspendisse nulla pretium, rhoncus tempor placerat fermentum, enim integer ad vestibulum volutpat. Nisl rhoncus turpis est, vel elit, congue wisi enim nunc ultricies sit, magna tincidunt. Maecenas aliquam maecenas ligula nostra.



Section 1

Untitled

LOREM IPSUM

1. **Lorem ipsum dolor sit amet**
2. **Consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.**
3. **Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.**
4. **Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.**

Integers, as you probably know, are numbers without any fractional part, ranging from -∞ to ∞. Floating point numbers on the other hand are numbers with fractional values, expressed on computers using decimal notation rather than fractions (in most cases). In Python and most other languages, if you do math with only integers, the result is also an integer, but if either value is floating point, the result is floating point.

Chapter 6

Making!

Lorem ipsum dolor sit amet, ligula suspendisse nulla pretium, rhoncus tempor placerat fermentum, enim integer ad vestibulum volutpat. Nisl rhoncus turpis est, vel elit, congue wisi enim nunc ultricies sit, magna tincidunt. Maecenas aliquam maecenas ligula nostra.

Chapter 7

A Compilation of Concepts

This chapter reviews all of the concepts introduced in the earlier chapters. It's a useful reference, and can be referred to when you're learning the concepts to see additional examples.

A Summary of Common Data Types

MAIN DATA TYPES IN PYTHON

1. **Integer**
2. **Floating Point Number**
3. **String**
4. **List / Tuple (immutable)**
5. **Dictionary**
6. **Set**

	Conversion Function	Mutable	Main Operators, Functions & Methods
Integer	int()	-	+, -, *, /, **, %
Floating Point Number	float()	-	+, -, *, /, **, % import math

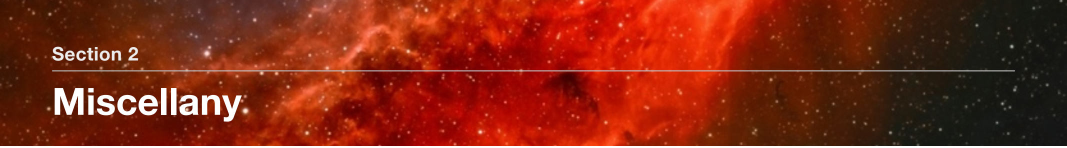
String	str()	No	+, *, [a:b]
			len()
			strip(), split(), find(), rfind(), replace()

List	list()	Yes	+, *, [a:b]
			len(), sorted()
			append(), remove(), count(), sort(), reverse()

Tuple	tuple()	No	+, *, [a:b]
			len(), sorted()
			count(), index()

Dictionary	dict()	Yes	[key]
			len()
			keys(), values(), items()

Set	set()	Yes	I, &, -, ^
			len()
			add(), remove(), clear(), union(), intersection(), difference()



Miscellany

“HI” program

```
from turtle import *

Turtle()

a=[90,180,90,90,180,90,0,180,-90,90,180]

fdr=[100,50,50,50,100,40,50,25,100,25,50]

ht()

clear()

for i in range(11):

    left(a[i])

    forward(fdr[i])
```