Chapter 3
Writing Simple Programs
Charles Severance

Textbook: Using Google App Engine, Charles Severance
What Is Programming?

- Within the web server we set lots and lots of “requests” which we need to respond to
- This is the request/response cycle
- We don’t want to do this by hand
- So we write a program to communicate how we want each of the requests to be handled
Software

Input Devices

Central Processing Unit

Main Memory

Output Devices

Secondary Memory

Generic Computer

What Next?
Program Steps or Program Flow

- Like a recipe or installation instructions, a program is a sequence of steps to be done in order.
- Some steps are conditional - they may be skipped.
- Sometimes a step or group of steps are to be repeated.
- Sometimes we store a set of steps to be reused over and over as needed several places throughout the program.
add 300 grams of flour
add 100 ml of milk
add an egg

if altitude > 2000:
    add an egg

add 30 grams of salt

while there are too many lumps:
    Beat mixture with a fork

open and add provided flavor packet
add 300 grams of flour
add 100 ml of milk
add an egg

if altitude > 2000:
    add an egg

add 30 grams of salt

while there are too many lumps:
    Beat mixture with a fork

open and add provided flavor packet
Elevators in Europe!
usf = input('Enter the US Floor Number: ')
wf = usf - 1
print 'Non-US Floor Number is',wf

python elev.py
Enter the US Floor Number: 2
Non-US Floor Number is 1
The Essence of Programming
Program Steps or Program Flow

• Like a recipe or installation instructions, a program is a sequence of steps to be done in order

• Some steps are conditional - they may be skipped

• Sometimes a step or group of steps are to be repeated

• Sometimes we store a set of steps to be reused over and over as needed several places throughout the program
Sequential Steps

Program:

```python
x = 1
print x
x = x + 1
print x
```

Output:

```
x = 1
print x
x = x + 1
print x
```

1
2

When a program is running, it flows from one step to the next. We as programmers set up “paths” for the program to follow.
Program:

```
x = 5
if x < 10:
    print 'Smaller'
if x > 20:
    print 'Bigger'
print 'Finis'
```

Output:

Smaller
Finis
Loops (repeated steps) have iteration variables that change each time through a loop. Often these iteration variables go through a sequence of numbers.

Program:
```python
for i in range(5):
    print i
    print 'Bye'
```

Output:
```
0
1
2
3
4
Bye
```
We call these little stored chunks of code “subprograms” or “functions”.

Program:

```python
def hello():
    print 'Hello'
    print 'Fun'

hello()

print 'Zip'

hello()
```

Output:

- Hello
- Fun
- Zip
- Hello
- Fun
print 'Your guess is', guess

answer = 42

if guess < answer :
    print 'Your guess is too low'

if guess == answer :
    print 'Congratulations!'  

if guess > answer :
    print 'Your guess is too high'
answer = 42

guess == answer

true

print 'Good Guess'

false

print 'Bad guess'

print 'Your guess is', guess

if guess == answer :
    print 'Good guess'
else:
    print 'Bad guess'
Nesting

- We can place a block of code within another block of code
- We call this “nesting” because the inner block is snugly nested within the outer block

```python
print 'Your guess is', guess

answer = 42

if guess == answer:
    print 'Congratulations!
else:
    if guess < answer:
        print 'Your guess is too low'
    else:
        print 'Your guess is too high'
```
```python
print 'Your guess is', guess

answer = 42

if guess == answer:
    print 'Congratulations!

elif guess < answer:
    print 'Your guess is too low'

else:
    print 'Your guess is too high'
```
Variables

- **Variables** are named locations in the computer’s main **memory**

- We programmers use **variables** to store values that we want to use later in the program

- We get to pick the names of our **variables** (within limits)

```python
usf = input('Enter the US Floor Number: ')  
wf = usf - 1  
print 'Non-US Floor Number is',wf
```
Rules for Python Variable Names

- Must start with a letter or underscore _
- Must consist of letters and numbers
- Case Sensitive

- Good:  x  usf  _food  food16  FOOD
- Bad:   42secret  :usf  value-7
- Different:  usf  Usf  USF
Menmonic Variables

• We often try to pick mnemonic variable names to help us remember what we intended as the contents of a variable

• Non-mnemonic: x42 xyzzy snagl1 a123 ljklljk

• Mnemonic: count lines word usf wf

```python
usf = input('Enter the US Floor Number: ')
wf = usf - 1
print 'Non-US Floor Number is',wf
```

http://en.wikipedia.org/wiki/Mnemonic
Mnemonic

```python
usf = input('Enter the US Floor Number: ')
wf = usf - 1
print 'Non-US Floor Number is',wf
```

Non-Mnemonic

```python
dsjdkjds = input('Enter the US Floor Number: ')
xsjdkjds = dsjdkjds - 1
print 'Non-US Floor Number is', xsjdkjds
```
Reserved Words

- You can not use reserved words as variable names / identifiers

    and del for is raise
    assert elif from lambda return
    break else global not try
    class except if or while
    continue exec import pass yield
    def finally in print
Assignment Statements

• variable = expression

• Evaluate the expression to a value and then put that value into the variable

  x = 1
  spam = 2 + 3
  spam = x + 1
  x = x + 1
Slow Motion Assignment

- We can use the **same variable** on the left and right side of an assignment statement
- Remember that the right side is evaluated *before* the variable is updated
Input Statements

• `input('Prompt')` - displays the prompt and waits for us to input an expression - this works for numbers

• In Chapter 4 we will see how to read strings

```python
>>> x = input('Enter ') Enter 123 >>> print x 123
```
We use the term “constant” or “literal” to indicate a value that is not a variable.

```python
usf = input('Enter the US Floor Number: ')
wf = usf - 1
print('Non-US Floor Number is', wf)
```
String Data

- Modern programs work with string data ('Fred', 'Ann Arbor', ...) far more often than numeric data (1, 2, 3.14159)
- Python has great support for working with strings
print 'Your guess is', guess

answer = 42

if guess == answer :
    msg = 'Congratulations!'
elif guess < answer :
    msg = 'Your guess is too low'
else:
    msg = 'Your guess is too high'

print msg

We can use string constants and string variables to simplify our program. Remember that a variable is a place in memory that we can use to store something that we want to use later in our program.
print 'Your guess is', guess

answer = 42

if guess < answer :
    print 'Your guess is too low'

if guess == answer :
    print 'Congratulations!'

if guess > answer :
    print 'Your guess is too high'

When you type “25” into the form and press “Submit”, your browser sends a string like “guess=25” to your web application.
Indexing Strings

• We can look at each character in a string by “indexing” the string with square brackets “[“ and “]”

• The first character in a string is [0]

```python
>>> txt = 'guess=25'
>>> print txt[0]
g
>>> print txt[1]
u
```
Slicing Strings

- We can extract a range of characters in a string using two numbers and a colon (:) in the square brackets.

- The second value means “up to but not including”.

```python
>>> txt = 'guess=25
01234567

python
>>> txt = 'guess=25'
>>> print txt[2:4]
es
>>> print txt[2:5]
ess
>>> 
```
Slicing Strings

- When we slice strings, we can omit the first or second number and it implies beginning and end of the string respectively.

```python
>>> txt = 'guess=25
>>> print txt[:5]
guess
01234
```

```python
>>> txt = 'guess=25
>>> print txt[6:]
25
```

01234567
Concatenating Strings

- We use the “+” operator to concatenate two strings.
- If we want a space between the strings, we need to add the space.

```python
>>> think = 'happy' + 'thoughts'
>>> print think
happythoughts

>>> think = 'happy' + ' ' + 'thoughts'
>>> print think
happy thoughts
```
The Python String Library

- Python has a number of powerful string manipulation capabilities in the string library (an example of the store and reuse pattern)

```python
txt = 'guess=25'
>>> print txt.find('=')
5
>>> print txt.find('pizza')
-1
```
The Python String Library

- Other capabilities in the string library include: \texttt{lowercase()}, \texttt{rfind()}, \texttt{split()}, \texttt{strip()}, \texttt{rstrip()}, \texttt{replace()}, and many more

\url{http://docs.python.org/library/stdtypes.html}
Types and Conversion

- Every variable and constant in Python as a "type" and Python knows the type.
- If you do something that is not allowed for a particular type, you will get an error.
- We can ask Python which type a variable is using the built-in type() function.

```python
>>> txt = 'guess=25'
>>> print type(txt)
<type 'str'>
>>> pos = txt.find('=')
>>> print pos
5
>>> print type(pos)
<type 'int'>
```
Breaking the Rules...

```python
>>> txt = 'abc' + 'def'
>>> print txt
abcdef
>>> num = 36 + 6
>>> print num
42
>>> huh = 'abc' + 6
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot concatenate 'str' and 'int' objects
```
Converting Integer to String

```python
>>> txt = 'abc' + 'def'
>>> print txt
abcdef

>>> num = 36 + 6
>>> print num
42

>>> huh = 'abc' + str(6)
>>> print huh
abc6
```

If we convert the integer to a string using the built-in `str()` function - the types match for the concatenation (+) operation.
Parsing a String

- Sometimes we want to break a string into pieces and do something with those pieces in several steps

```
>>> txt = 'guess=25
>>> pos = txt.find('=')
>>> sub = txt[pos+1:]
>>> print sub
25
>>> print type(sub)
<type 'str'>
>>> ival = int(sub)
>>> print ival
25
>>> print type(ival)
<type 'int'>
```
Multi-Value Variables: Collections

- So far all variables we have seen contain a single value and if we put a new value into the variable, it over-writes the existing value

- Collections are a type of variable that can contain more than one value at the same time

- We need a way to organize, store, and retrieve these multiple values
A Python List Object

- A Python `list()` object can contain more than one item
- Lists are stored in order and are indexed by the position of a value within a list
- They are like a one column spreadsheet
- Python lists follow “European Elevator” rules
We create an empty list by calling the built-in function `list()`

We add new elements using `append()`

We can find the length of the list using the built-in function `len()`

We can index the list with square brackets

```python
>>> pals = list()
>>> pals.append('Glenn')
>>> pals.append('Sally')
>>> pals.append('Joe')
>>> print pals
['Glenn', 'Sally', 'Joe']
>>> print type(pals)
<type 'list'>
>>> print len(pals)
3
>>> print pals[0]
Glenn
>>> print pals[2]
Joe
```
We can replace an element in a list by using an index in an assignment statement.

We can sort a list using the sort method in the list library.

```python
>>> print pals
['Glenn', 'Sally', 'Joe']
>>> pals[2] = 'Joseph'
>>> print pals
['Glenn', 'Sally', 'Joseph']
>>> pals.sort()
>>> print pals
['Glenn', 'Joseph', 'Sally']
>>> print pals
['Glenn', 'Joseph', 'Sally']
```
Loops are an example of the “repeated code” pattern

We construct a loop using `for` and `in` will execute a block of code once for each value in the list

We define an iteration variable that takes on the successive elements of the list each time through the loop

```python
pals = ['Glenn', 'Sally', 'Joseph']
for x in pals:
    print x
print 'Done'
```
Loops are an example of the “repeated code” pattern.

We construct a loop using `for` and `in` will execute a block of code once for each value in the list.

We define an iteration variable that takes on the successive elements of the list each time through the loop.

```python
>>> print pals
['Glenn', 'Sally', 'Joseph']
>>> for x in pals:
...     print x
...
Glenn
Sally
Joseph
>>>
Looping Through Strings

- Strings function very much like a “list of characters”

- So we can construct a loop using `for` and `in` will execute a block of code once for each value in the list

- We define an iteration variable that takes on the successive elements of the list each time through the loop

```python
>>> txt = 'guess=25'
>>> for x in txt:
...     print x
...
$$guess = 25$$
```
Python’s Backpack: Dictionaries

• Sometimes we want a bunch of stuff in a collection where each item has a label and the label allows us to store and retrieve a value

• It is more like a two-column spreadsheet

<table>
<thead>
<tr>
<th>label</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>Glenn</td>
</tr>
<tr>
<td>last</td>
<td>Golden</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:gleeng@umich.edu">gleeng@umich.edu</a></td>
</tr>
<tr>
<td>phone</td>
<td>517-303-8700</td>
</tr>
</tbody>
</table>
• We create an empty dictionary by calling `dict()`

• We fill up our dictionary with assignment statements where the index is the "key" or "label" which marks a value

• When we print a dictionary we see a list of mappings of a key to a value

```python
>>> pal = dict()
>>> pal['first'] = 'Glenn'
>>> pal['last'] = 'Golden'
>>> pal['email'] = 'glenng@umich.edu'
>>> pal['phone'] = '517-303-8700'
>>> print pal
{'phone': '517-303-8700', 'last': 'Golden', 'email': 'glenng@umich.edu', 'first': 'Glenn'}
```

<table>
<thead>
<tr>
<th>label</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>Glenn</td>
</tr>
<tr>
<td>last</td>
<td>Golden</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:glenng@umich.edu">glenng@umich.edu</a></td>
</tr>
<tr>
<td>phone</td>
<td>517-303-8700</td>
</tr>
</tbody>
</table>
Retrieving Data

- To retrieve an *value* from the dictionary, we can use the index operation “[“

- But we must make sure that the *key* exists

<table>
<thead>
<tr>
<th>label</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>Glenn</td>
</tr>
<tr>
<td>last</td>
<td>Golden</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:gleeng@umich.edu">gleeng@umich.edu</a></td>
</tr>
<tr>
<td>phone</td>
<td>517-303-8700</td>
</tr>
</tbody>
</table>

```python
>>> print pal
{
'phone': '517-303-8700',
'last': 'Golden',
'email': 'glenng@umich.edu',
'first': 'Glenn'
}
```

```python
>>> print pal['phone']
517-303-8700
```

```python
>>> print pal['age']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'age'
```
Safe Dictionary Retrieval

- We can deal with the missing key by using the built-in function `get()` and specifying a value to return if the key is not found.

```python
>>> print pal
{'phone': '517-303-8700', 'last': 'Golden', 'email': 'glenng@umich.edu', 'first': 'Glenn'}
>>> print pal.get('age','Age not available')
Age not available
>>> print pal.get('phone', 'Phone not available')
517-303-8700
```
Looping Through a Dictionary

- **Dictionaries** do not maintain order - but we can loop through them.

- We construct a loop with `for` and `in`.

- The *iteration variable* loops through the *keys* in the *dictionary*.

```python
>>> print pal
{'phone': '517-303-8700', 'last': 'Golden', 'email': 'glenng@umich.edu', 'first': 'Glenn'}
>>> for z in pal:
    ...    print z
    ...    phone
    ...    last
    ...    email
    ...    first
>>> 
```
Looping Through a Dictionary

- If we can loop through the **keys**, we can then use those **keys** to look up the **values**.

<table>
<thead>
<tr>
<th>key</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>first</td>
<td>Glenn</td>
</tr>
<tr>
<td>last</td>
<td>Golden</td>
</tr>
<tr>
<td>email</td>
<td><a href="mailto:gleeng@umich.edu">gleeng@umich.edu</a></td>
</tr>
<tr>
<td>phone</td>
<td>517-303-8700</td>
</tr>
</tbody>
</table>

```python
>>> print pal
{'phone': '517-303-8700', 'last': 'Golden', 'email': 'glenng@umich.edu', 'first': 'Glenn'}

>>> for key in pal:
...    print key, pal[key]
...
...    phone 517-303-8700
...    last Golden
...    email gleeng@umich.edu
...    first Glenn

>>>```
Store/Reuse: Functions

- In the last pattern, we want to write code once and reuse it several places - this allows us to make changes one place and allows us to organize larger programs into logical sub-units.
Function: How Long is a Sequence?

- The `len()` function takes a string as a parameter and returns the number of characters in the string.
- Actually `len()` tells us the number of elements of any set or sequence.

```python
>>> greet = 'Hello Bob'
>>> print len(greet)
9
>>> x = [ 1, 2, 'fred', 99]
>>> print len(x)
4
>>> 
```
Len Function

A function is some stored code that we use. A function takes some input parameter(s) and produces an output return value.

Guido wrote this code

```python
>>> greet = 'Hello Bob'
>>> x = len(greet)
>>> print x
9
```
A function is some stored code that we use. A function takes some input parameter(s) and produces an output return value.

```python
>>> greet = 'Hello Bob'
>>> x = len(greet)
>>> print(x)
9

'reHello Bob' (a string)
def len(inp):
    blah
    blah
    for x in y:
        blah
        blah

9 (a number)
A Trivial Function

- The `def` keyword indicates the beginning of a block and defines a name for the function.
- The block of code is not executed as part of the “def” process.
- We can execute the code later using the name we assigned to the function.

```python
>>> def welcome():
...     print 'Hello'
...
>>> welcome()
Hello
```

```python
>>> def welcome():
...     print 'Hello'
...
>>> welcome()
Hello
```
Parameters to Functions

• Sometimes we want to feed the function some value (i.e. a parameter) as its input so we can use the function for different purposes

• We define the “formal parameter” on the def statement

• We pass in the “actual parameter” on the function call

```python
>>> def welcome(name):
...     print 'Hello', name
...     ...

>>> welcome('Glenn')
Hello Glenn

>>> welcome('Sally')
Hello Sally
```

Return Values

• A function can “return” a value back to its caller using the “return” statement

• This return value “comes back” and can be used in an assignment statement or expression

>>> def greet(lang):
...     if lang == 'es':
...         return 'Hola'
...     elif lang == 'fr':
...         return 'Bonjour'
...     else:
...         return 'Hello'
...

>>> xgr = greet('fr')
>>> print xgr, 'Michael'
Bonjour Michael
>>>
Return Values

- A function can “return” a value back to its caller using the “return” statement.

- This return value “comes back” and can be used in an assignment statement or expression.

```python
>>> def greet(lang):
...     if lang == 'es':
...         return 'Hola'
...     elif lang == 'fr':
...         return 'Bonjour'
...     else:
...         return 'Hello'
...     
...     print(greet('en')),'Glenn'
Hello Glenn
>>> print(greet('es')),'Sally'
Hola Sally
>>> print(greet('fr')),'Michael'
Bonjour Michael
>>> 
```
The **try** / **except** Structure

- Sometimes we know a line of code may cause traceback - perhaps because it is processing user input that may be flawed

- You surround a **dangerous section** of code with **try** and **except**.

- If the code in the **try** works - the **except** is skipped

- If the code in the **try** fails - it jumps to the **except** section
$ cat notry.py
astr = 'fourtytwo'
istr = int(astr)
print 'First', istr
astr = '123'
istr = int(astr)
print 'Second', istr

$ python notry.py
Traceback (most recent call last):
  File "notry.py", line 6, in <module>
    istr = int(astr)
ValueError: invalid literal for int() with base 10: 'fourtytewo'

All Done
Generic Computer

Software

Input Devices

Central Processing Unit

Output Devices

Main Memory

Secondary Memory
$ cat tryexcept.py
astr = 'fourtytwo'
try:
    istr = int(astr)
except:
    istr = -1
print 'First', istr
astr = '42'
try:
    istr = int(astr)
except:
    istr = -1
print 'Second', istr

When the first conversion fails - it just drops into the except: clause and the program continues.

$ python tryexcept.py
First -1
Second 42

When the second conversion succeeds - it just skips the except: clause and the program continues.
try / except

astr = 'fourtytwo'
try:
    print 'Hello'
    istr = int(astr)
    print 'There'
except:
    istr = -1
print 'Done', istr
Comments in Python

- Python ignores blank lines in Python programs and ignores everything after it sees “#” on a line

```python
# This program helps travelers use elevators
usf = input('Enter the US Floor Number: ')  
wf = usf - 1  # The conversion is quite simple
print 'Non-US Floor Number is', wf
```
Comments in Python

- Python ignores blank lines in Python programs and ignores everything after it sees ‘#’ on a line

```python
usf = input('Enter the US Floor Number: ')
wf = usf - 1
print 'Non-US Floor Number is',wf
```
Questions...