





#### **Answers to Selected Exercises**

## Chapter 2

```
5. Loops and numbers
```

```
a)
    i = 0
    while i < 11:
        i += 1

b)
    for i in range(11):
        pass
6. Conditionals</pre>
```

```
n = int(raw_input('enter a number: '))
if n < 0:
    print 'negative'
elif n > 0:
    print 'positive'
else:
    print 'zero'
```

www.aibbt.com 让未来触手可及









#### Appendix A

```
7. Loops and strings
       s = raw_input('enter a string: ')
       for eachChar in s:
            print eachChar
                               # (does not print index)
  or
       for i in range(len(s)):
            print i, s[i]
  or
       i = 0
       slen = len(s)
       while i < slen:</pre>
            print i, s[i]
  or
       for i, x in enumerate(s):
            print i, x
8. Loops and operators
       subtot = 0
       for i in range(5):
            subtot += int(raw_input('enter a number: '))
       print subtot
       # uses sum() BIF and generator expressions
       print sum(int(raw_input('enter a number: ')) for i in range(5))
```

## Chapter 3

4. Statements

Use;

5. Statements

Use \ (unless part of a comma-separated sequence in which case \ is optional)

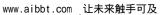
7. Identifiers

```
40XL
                  number
                  symbol
$saving$
print
                  keyword
0x40L
                  number
big-daddy
                  symbol
2hot2touch
                  number
thisIsn'tAVar
                  symbol
if
                  keyword
counter-1
                  symbol
```









1013



## Chapter 4

#### 1. Python objects

All Python objects have three attributes: type, ID, and value. All are readonly with a possible exception of the value (which can be changed only if the object is mutable).

#### 5. str() vs. repr()

repr() is a built-in function while str() was a built-in function that changed to a factory function in Python 2.2. They will both return a string representation of an object; however, str() returns a *printable* string representation while repr() (and the backquote operator ``) return an *evaluatable* string representation of an object, meaning that it is a string that represents a (valid) Python object that would be created if passed to eval().

#### 6. Object equality

Since there exists only one (type) object for each built-in type, there is no need to check their values; hence, only the latter form should be used.

#### Chapter 5

#### 8. Geometry

```
import math

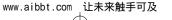
def sqcube():
    s = float(raw_input('enter length of one side: '))
    print 'the area is:', s ** 2., '(units squared)'
    print 'the volume is:', s ** 3., '(cubic units)'

def cirsph():
    r = float(raw_input('enter length of radius: '))
    print 'the area is:', math.pi * (r ** 2.),
    '(units squared)'
    print 'the volume is:', (4. / 3.) * math.pi * (r ** 3.), '(cubic units)'

sqcube()
cirsph()
```









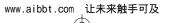




#### 1014 Appendix A

```
11. Modulus
   a)
        for i in range(0, 22, 2): # range(0, 21, 2) okay too
             print i
   or
        for i in range(22):
                                            # range(21) okay too
             if i % 2 == 0:
                 print i
   b)
        for i in range(1, 20, 2):
                                            # range(1, 21, 2) okay too
             print i
   or
        for i in range(20):
                                            # range(21) okay too
             if i % 2 != 0:
                 print i
   c)
        When i % 2 == 0, it's even (divisible by 2), otherwise it's odd.
 Chapter 6
 1. Strings
         find(), rfind(), index(), rindex(); can also use the in
        operator.
2. Identifiers
        import string
        alphas = string.letters + '_'
        alnums = alphas + string.digits
        iden = raw_input('Identifier to check? ')
        if len(iden) > 0:
              if iden[0] not in alphas:
                  print "Error: first char must be alphabetic"
              else:
                   if len(iden) > 1:
                        for eachChar in iden[1:]:
                              \textbf{if} \ \texttt{eachChar} \ \textbf{not} \ \textbf{in} \ \texttt{alnums:}
                                   print "Error: others must be alnum"
                                   break
                        else:
```

print 'Error: keyword name'
else:
 print 'Error: no identifier entered'



import keyword

print 'ok'

if iden not in keyword.kwlist:









#### **Answers to Selected Exercises**

1015

## Chapter 7

1. Dictionary methods dict.update()

3. Dictionary methods

```
a)
    keys = dict.keys()
    keys.sort()

or
    sorted(dict.keys())
```

4. Creating dictionaries

```
# assumes and list2 are the same length
d = {}
for i in range(len(list1)):
    d[list1[i]] = list2[i]

or
d = {}
for i, x in enumerate(list1):
    d[x] = list2[i]

or
d = dict(map(None, list1, list2))

or
d = dict(zip(list1, list2))
```

7. Inverting dictionaries

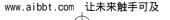
```
list1 = oldDict.values()
list2 = oldDict.keys()
```

Now apply the solutions to Problem 4.

Note that these solutions are destructive, meaning that for one-to-many dictionaries, keys that share the same values will only have the latest installed value for the value that is now a key. Extra Credit: Come up with a non-destructive solution where keys that share the same values in the old dictionary are now stored inside a list as the value for the corresponding key in the new dictionary.













#### Appendix A

## Chapter 8

```
3. range() built-in function
  a)
       range(10)
4. Prime numbers
       import math
       def isprime(num):
            count = int(math.sqrt(num))
           while count > 1:
                if num % count == 0:
                     return False
                count -= 1
            else:
                return True
```

## Chapter 9

```
2. File access
```

```
f = open(raw_input('enter filename: '))
num = int(raw_input('enter number of lines: '))
for eachLine in f:
    if i == num:
        break
                         # suppress NEWLINE
    print eachLine,
    i += 1
f.close()
```

# argv

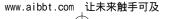
13. Command-line arguments

```
b)
    import sys
    print "# of args", len(sys.argv)
                                     # argc
    print "args:", sys.argv
```

## Chapter 10

- 1. Raising exceptions
- 2. Raising exceptions d)













#### **Answers to Selected Exercises**

4. Keywords

try-except monitors the try clause for exceptions and execution jumps to the matching except clause. However, the **finally** clause of a try-finally will be executed regardless of whether or not an exception occurred. How does the try-except-finally statement work?

- 5. Exceptions (we'll provide the solution, but you have to determine why):
  - a) SyntaxError
  - b) IndexError
  - c) NameError
  - d) ZeroDivisionError
  - e) ValueError
  - f) TypeError

## Chapter II

2. Functions

```
def sumtimes(x, y):
    return (x+y, x*y)
```

6. Variable-length arguments

```
def printf(string, *args):
    print string % args
```

#### Chapter 12

2. Importing attributes

```
a)

import mymodule ⇒ mymodule.foo()

and

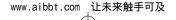
from mymodule import foo ⇒ foo()
```

If you use the **import** statement, the module name is brought into the local namespace, and foo() is only accessible from the module's namespace.

If you use the **from-import** statement, "foo()" itself is brought into the local namespace. In this case, you do not need to use the module's namespace to access it.













#### Appendix A

## Chapter 13

2. Functions versus methods

Methods are basically functions but tied to a specific class object type. They are defined as part of a class and are executed as part of an instance of that class.

15. Delegation

It makes no difference whether we use open() or capOpen() to read our file because in capOpen.py, we delegated all of the reading functionality to the Python system defaults, meaning that no special action is ever taken on reads. The same code would be executed, i.e., none of read(), readline(), or readlines() are overridden with any special functionality.

## Chapter 14

1. Callable objects

Functions, methods, classes, callable class instances

3. input() vs. raw\_input()
 raw\_input() returns user input as a string; input() returns the evaluation
 of the user input as a Python expression. In other words:

## Chapter 15

Regular expressions

 Matching strings bat, hat, bit, etc.

[bh][aiu]t

2. First name last

$$[A-Za-z-]+[A-Za-z-]+$$

(Any pair of words separated by a single space, e.g., first and last names, hyphens allowed)

3. Last name first

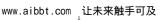
$$[A-Za-z-]+$$
,  $[A-Za-z]$ 

(Any word and single letter separated by a comma and single space, e.g., last name, first initial)

$$[A-Za-z-]+, [A-Za-z-]+$$











## •



(Any pair of words separated by a comma and single space, e.g., last, first names, hyphens allowed)

8. Python longs

d+[lL]

(Decimal [base 10] integers only)

9. Python floats

$$[0-9]+(\.[0-9]*)?$$

(Describes a simple floating point number, that is, any number of digits followed optionally by a single decimal point and zero or more numeric digits, as in "0.004," "2," "75.," etc.)

## Chapter 16

3. Sockets

**TCP** 

6. Daytime service

```
>>> import socket
>>> socket.getservbyname('daytime', 'udp')
13
```

#### Chapter 17

20. Identifiers

**pass** is a keyword, so it cannot be used as an identifier. The common idiom in all such cases is to append an underscore ( \_ ) to the name of the offending variable.

## Chapter 18

2. Python threads

I/O-bound . . . why?

## Chapter 19

1. Client/server architecture

Window(ing) clients are GUI events generated usually by users which must be processed by the window(ing) system that acts as the server; it is responsible for making timely updates to the display as to be apparent to the user.











#### Appendix A

## Chapter 20

#### 15. CGI errors

The Web server returns either no data or error text, which results in an HTTP 500 or Internal Server Error in your browser because that (returned data) is not valid HTTP or HTML data. The cgitb module captures the Python traceback and returns it as valid data through CGI, which gets displayed to the user . . . a great debugging tool.

## Chapter 21

- 1. Extending Python
  - Performance improvement
  - Protecting source code
  - New or desired change of functionality
  - And more!

#### Chapter 22

#### 1. DB-API

The DB-API is a common interface specification for all Python database adapters. It is good in that it forces all adapter writers to code to the same specification so that end-user programmers can write consistent code that can be (more) easily ported to other databases with the minimum amount of effort.

## Chapter 23

3. Web services and the csv module

Replace the **for** loop in stock.py with the following:

```
import csv
for tick, price, chg, per in csv.reader(f):
    print tick.ljust(7), ('%.2f' % round(float(price),
2)).rjust(6), chg.rjust(6), per.rjust(6)
```





