

---

# Data Structures and Algorithms in Python

**Michael T. Goodrich**

Department of Computer Science  
University of California, Irvine

**Roberto Tamassia**

Department of Computer Science  
Brown University

**Michael H. Goldwasser**

Department of Mathematics and Computer Science  
Saint Louis University

---

## Instructor's Solutions Manual

WILEY

# Chapter

# 1

## Python Primer

### Hints and Solutions

#### Reinforcement

**R-1.1) Hint** The modulo operator could be useful here.

**R-1.1) Solution**

```
def is_multiple(n, m):  
    return n % m == 0
```

**R-1.2) Hint** Use bit operations.

**R-1.2) Solution**

```
def is_even(k):  
    return (k & 1 == 0)
```

**R-1.3) Hint** Keep track of the smallest and largest value while looping.

**R-1.3) Solution**

```
def minmax(data):  
    small = big = data[0]    % assuming nonempty  
    for val in data:  
        if val < small:  
            small = val  
        if val > big:  
            big = val  
    return small, big
```

**R-1.4) Hint** Although there is a formula for this, the easy thing to do is to write a loop.

**R-1.4) Solution**

```
def sum_of_squares(n):  
    total = 0  
    for j in range(1, n+1):  
        total += j*j  
    return total
```

**R-1.5) Hint** How can you describe the range of integers for the sum?

**R-1.5) Solution**

```
total = sum(j*j for j in range(1, n+1))
```

**R-1.6) Hint** Consider modifying the range over which you loop.

**R-1.6) Solution**

```
def sum_of_squares(n):
    total = 0
    for j in range(1, n+1, 2):
        total += j*j
    return total
```

**R-1.7) Hint** How can you describe the range of integers for the sum?.

**R-1.7) Solution**

```
total = sum(j*j for j in range(1, n+1, 2))
```

**R-1.8) Hint** Give your answer in terms of  $n$  and  $k$ .

**R-1.8) Solution**  $n+k$

**R-1.9) Hint** Where does the sequence start and end? What is the step size?

**R-1.9) Solution** `range(50,81,10)`

**R-1.10) Hint** Use a negative step size.

**R-1.10) Solution** `range(8, -10, -2)`

**R-1.11) Hint** Those look like powers of two!

**R-1.11) Solution** `[2**k for k in range(9)]`

**R-1.12) Hint** Use `randrange` to pick the index of the chosen element.

**R-1.12) Solution**

```
def choice(data):
    return data[randrange(len(data))]
```

---

## Creativity

**C-1.13) Hint** The Python function does not need to be passed the value of  $n$  as an argument.

**C-1.14) Hint** Note that both numbers in the pair must be odd.

**C-1.14) Solution**

```
def has_odd_pair(data):
    count=0
    for j in range(len(data)):
        if data[j] % 2 == 1:
            count++
        if count == 2:
            return True
    return False
```

**C-1.15) Hint** The simple solution just checks each number against every other one, but we will discuss better solutions later in the book. But make sure you don't compare a number to itself.

**C-1.15) Solution**

```
def distinct(data):
    for k in range(1, len(data)):
        for j in range(k):
            if data[j] == data[k]:
                return False
    return True
```

**C-1.16) Hint** Think about the semantics of `data[j] = data[j] * factor`.

**C-1.17) Hint** Try it out and see if it works!

**C-1.17) Solution** This does not work because it reassigns the value of local variable `val`, but not the entries of the list `data`.

**C-1.18) Hint** What are the factors of each number?

**C-1.18) Solution** `[k*(k+1) for k in range(10)]`

**C-1.19) Hint** Use the `chr` function with appropriate range

**C-1.19) Solution** `[chr(k) for k in range(97,123)]`

**C-1.20) Hint** Consider randomly swapping an element to the first position, then randomly swapping a remaining element to the second position, and so on.

**C-1.21) Hint** Use a list to store all the lines.

**C-1.21) Solution**

```
lines = [ ]
while True:
    try:
        single = input()
        lines.append(single)
    except EOFError:
        break # leave the while loop

print('\n'.join(reversed(lines)))
```

**C-1.22) Hint** Go back to the definition of dot product and write a for loop that matches it.

**C-1.22) Solution**

```
return [a[k]*b[k] for k in range(n)]
```

**C-1.23) Hint** Use a try-except structure.

**C-1.23) Solution**

```
try:
    data[k] = val
except IndexError:
    print("Don't try buffer overflow attacks in Python!")
```

**C-1.24) Hint** You can use the condition `ch in 'aeiou'` to test if a character is a vowel.

**C-1.24) Solution**

```
def num_vowels(text):
    total = 0
    for ch in text.lower():
        if ch in 'aeiou':
            total += 1
    return total
```

**C-1.25) Hint** Consider each character one at a time.

**C-1.26) Hint** Try a case analysis for each pair of integers and an operator.

**C-1.27) Hint** Either buffer the bigger value from each pair of factors, or repeat the loop in reverse to avoid the buffer.

**C-1.27) Solution**

```
def factors(n):                                # generator that computes factors
    buffer = [ ]
    k = 1
    while k * k < n:                            # while k < sqrt(n)
        if n % k == 0:
            yield k
            buffer.append(n // k)
        k += 1
    if k * k == n:                              # special case if n is perfect square
        yield k
    for val in reversed(buffer):
        yield val
```

**C-1.28) Hint** Use the `**` operator to compute powers.

**C-1.28) Solution**

```
def norm(v, p=2):
    temp = sum(val**p for val in v)
    return temp ** (1/p)
```

## Projects

**P-1.29) Hint** There are many solutions. If you know about recursion, the easiest solution uses this technique. Otherwise, consider using a list to hold solutions. If this still seems too hard, then consider using six nested loops (but avoid repeating characters and make sure you allow all string lengths).

**P-1.29) Solution** Here is a possible solution:

```
def permute(bag, permutation):
    # When the bag is empty, a full permutation exists
    if len(bag) == 0:
        print(' '.join(permutation))
    else:
        # For each element left in the bag
        for k in range(len(bag)):
            # Take the element out of the bag and put it at the end of the permutation
            permutation.append(bag.pop(k))

            # Permute the rest of the bag (recursively)
            permute(bag, permutation);

            # Take the element off the permutation and put it back in the bag
            bag.insert(k, permutation.pop())

permute(list('catdog'), [ ])
```

**P-1.30) Hint** This is the same as the logarithm, but you can use recursion here rather than calling the `log` function.

**P-1.31) Hint** While not always optimal, you can design your algorithm so that it always returns the largest coin possible until the value of the change is met.

**P-1.32) Hint** Do a case analysis to categorize each line of input.

**P-1.33) Hint** Write your program to loop continually until a quit operation is entered. In each iteration, collect a sequence of button pushes, and then output the result from processing that sequence of pushes.

**P-1.34) Hint** Define a way of indexing all the sentences and the location in each one and then work out a way of picking eight of these locations for a typo.

**P-1.35) Hint** Use a two-dimensional list to keep track of the statistics and a one-dimensional list for each experiment.

**P-1.36) Hint** You need some way of telling when you have seen the same word you have before. Feel free to just search through your list of words to do this here.