

## recursion

```
GCD
def rec_find_gcd(a, b):
    if a%b == 0:
        return b
    return rec_find_gcd(b, a%b)

reverse string
def reverse_string(s):
    if len(s) <= 1:
        return s
    return s[-1]+reverse_string(s[:-1])

recursive function returning a
tuple with quotient and remainder
of 2 integers
def rec_div(a, b):
    if a < b:
        return 0, a
    new_tuple = rec_div(a-b, b)
    return new_tuple[0]+1,
new_tuple[1]

takes in string of positive numbers
and letters and returns a string of
all the numbers in order
def rec_num_find(s):
    if len(s) == 0:
        return ''
    if s[0].isdigit():
        return s[0] +
rec_num_find(s[1:])
    else:
        return rec_num_find(s[1:])

OR
def rec_num_find2(s):
    if len(s) <= 1:
        if s.isdigit():
```

## recursion (cont)

```
        return s
        return ''
    a = rec_num_find2(s[:
len(s)/2])
    b = rec_num_find2(s[len(s)/2
:])
    return a + b

Takes a string and a pattern and
determines if pattern is in
string(no in operator)
def rec_detect(s, pat):
    if len(pat) > len(s):
        return False
    return pat==s[:len(pat)] or
rec_detect(s[1:], pat)

counts periods in the string
argument
def rec_period(s):
    if len(s) == 0:
        return 0
    if s[0] == '.':
        return 1 +
rec_period(s[1:])
    return rec_period(s[1:])

takes a string of characters and
prints all combinations = original
length
def rec_all_strings(s, result=''):
    if len(result) == len(s):
        print result
        return None
    for char in s:
        rec_all_strings(s,
result+char)
```

## Recursion Continued

```
power function that raises a to the
power of b
def power(a, b):
    if b == 0:
        return 1
    return a * power(a, b-1)

function that takes an integer and
returns binary representation in
list
def bin_rep(n):
    if n <= 1:
        return [n]
    return bin_rep(n/2) + [n % 2]

takes in tuple, returns set of all
possible tuples using the original
elements
def power_set(atuple):
    if len(atuple) == 0:
        return {atuple}
    temp = power_set(atuple[1:])
    result = set()
    result.update(temp)
    for item in temp:
        new_tuple = (atuple[0], )
+ item
        result.add(new_tuple)
    return result

super digit is sum of digits
continuously until it is a single
number ie. 9875 = 2
def super_digit(n):
    if n < 10:
        return n
    temp = 0
    for digit in str(n):
```



### Recursion Continued (cont)

```
temp = temp + int(digit)
return super_digit(temp)
```

### Basic Recursion

```
def recursive_sum(n):
    if n == 1:
        return 1
    return n + recursive_sum(n-1)
def recursive_factorial(n):
    if n == 0:
        return 1
    return n*recursive_sum(n-1)
def multiplication(x,n):
    if n == 0:
        return 0
    return x + multiplication(x, n-1)
def reverse_string(s):
    if len(s) <= 1:
        return s
    return s[-1]+reverse_string(s[:-1])
def is_palindrome(s):
    s=s.lower()
    if len(s)<=1:
        return True
    return s[0] == s[-1] and
is_palindrome(s[1:-1])
def min_list(list1):
    if len(list1) == 1:
        return list1[0]
    min_rest = min_list[1:]
    if min_list[0] < min_rest:
        return list1[0]
```

### Basic Recursion (cont)

```
return min_rest
def product(list1):
    if len(list1) == 1:
        return list1[0]
    return list1[0] *
product(list1[1:])
def rotate_right(list1, n):
    if n == 0 or len(list1) == 0:
        return list1
    list1.insert(0,list1.pop())
    return rotate_right(list1, n-1)
def rec_flatten(nested_list):
    if len(nested_list) == 1:
        return nested_list[0]
    return nested_list[0] +
rec_flatten(nested_list[1:])
def rec_fib(n):
    if n<=1:
        return n
    return rec_fib(n-1) +
rec_fib(n-2)
def rec_fib_efficient(n, d):
    #use a dictionary to store
values in the sequence
    if n in d:
        return d[n]
    x = rec_fib_efficient(n-q, d)
+rec_fib_efficient(n-2, d)
    d[n] = x
    return x
```

### Tuples

Defining a tuple with one value  
singleton = (3,)  
Tuple packing  
=a, b... or return a, b...  
Tuple Operators  
+, \*, 'in'  
Tuple Methods  
tuple.count(a) will count the number of times a is in the tuple  
tuple.index(a) will give the index of a  
**for i in range(len(tuple))** iterates over the *indices* of the items  
thus you will  
**print tuple[i]**  
**for i in tuple** will iterate over the items directly  
thus you will  
**print i**  
sequence unpacking  
t = (1, 10, 100)  
first, second, third = t

### Dictionaries

dictionary = dict() OR dictionary = {}  
to add key-value.... dictionary[12345] = 'John'  
**dictionary[12345]** will print 'John'  
**Operators**  
in  
get retrieves value associated with a certain key(returns None if not in dictionary)  
**Methods**



By **kwo**  
[cheatography.com/kwo/](https://cheatography.com/kwo/)

Not published yet.  
Last updated 14th November, 2016.  
Page 2 of 4.

Sponsored by **ApolloPad.com**  
Everyone has a novel in them. Finish Yours!  
<https://apollopad.com>

## Dictionaries (cont)

get dictionary.get(key) gives the value associated with key  
keys dictionary.keys() gives all keys  
values dictionary.values() gives all values  
items dictionary.items() gives a list of key-value pairs

## Dictionary Code

### Reverse Lookup

```
def find_key(d, value_to_find):
    for key in d:
        if d[key] ==
value_to_find:
            return key
    return None
```

### grouping grades

```
grade_list = [100, 98, 76, 65, 61,
80, 75, 96, 90, 67, 87]
hist = {}
for i in range(1, 5):
    hist['bucket '+str(i)] = 0

for grade in grade_list:
    if grade >= 90:
        hist['bucket 1'] += 1
    elif grade >= 80:
        hist['bucket 2'] += 1
    elif grade >= 70:
        hist['bucket 3'] += 1
    else:
        hist['bucket 4'] += 1
for i in range(1, 5):
```

## Dictionary Code (cont)

```
print 'There are',
hist['bucket '+str(i)], 'grades in
bucket', i
```

## Sets

Set is an unordered collection of unique elements. All elements are immutable  
defining an empty set  
empty = set()  
using {} gives a dict  
sets are good for fast membership testing regardless of how many elements are in the set  
IE. set1 = {'BC', 'BU', 'NEU', 'BC', 'BC'}  
len(set1)  
3  
Set methods  
set.add(a) adds element a to the set  
red.intersection(blue) gives the intersection of blue and red  
red.update(blue) updates red with the union of itself and blue  
red -= {1, 2} is set difference. It is set red - 1 and 2

## Search Algorithms

```
def lin_search_general(input_list,
value):
    """Implementation of the Linear
Search Algorithm for an arbitrary
list
    Determine whether a given
value (number) exists in a list of
numbers that is ordered in an
ascending (increasing) fashion.
```

## Search Algorithms (cont)

```
Input arguments input_list--any
list of numbers (sorted or not
sorted)value--any number"""
    if len(input_list) == 0:
        return False
    for i in
range(len(input_list)):
        if value == input_list[i]:
            return True
        return False
def lin_search(ordered_list,
value):
    """Implementation of the Linear
Search Algorithm.
    Input arguments ordered_list--
any list of numbers that is sorted
in an ascending fashion value--any
number"""
    if len(ordered_list)==0 or
ordered_list[-1]<value:
        return False
    for i in
range(len(ordered_list)):
        if value ==
ordered_list[i]:
            return True
        if value <
ordered_list[i]:
            return False
        return False
def binary_search(ordered_list,
value):
    """Implementation of the Binary
Search Algorithm.
    Determine whether a given value
(number) exists in a list of
numbers
    that is ordered in an ascending
(increasing) fashion.
    Input arguments
    ordered_list--any list of
numbers that is sorted in an
ascending fashion
```

## Search Algorithms (cont)

```

value--any number"""
    if len(ordered_list)==0 or
ordered_list[-1]<value or
value<ordered_list[0]:
        return False
    low = 0
    high = len(ordered_list) - 1
    while low <= high:
        mid = (low + high) / 2
        if value ==
ordered_list[mid]:
            return True
        if value <
ordered_list[mid]:
            high = mid - 1
        else:
            low = mid + 1
    return False
def
binary_search_rec(ordered_list,
value):
    if len(ordered_list)==0 or
value<ordered_list[0] or
ordered_list[-1]<value:
        return False
    return
binary_search_helper(ordered_list,
value)
def
binary_search_helper(ordered_list,
value):
    if len(ordered_list) == 0:
        return False
    mid = (len(ordered_list)-1) /
2
    if value == ordered_list[mid]:
        return True
    if value < ordered_list[mid]:
        return
binary_search_helper(ordered_list[:
mid], value)

```

## Search Algorithms (cont)

```

return
binary_search_helper(ordered_list[m
id+1:], value)

```

## Sort Algorithms

```

def selection_sort(in_list):
    """Implementation of the
Selection Sort algorithm (in-
place)."""
    n = len(in_list)
    for i in range(n-1):
        # finding the min value in
a portion of the list
        min_value = in_list[i]
        min_index = i
        for j in range(i+1, n):
            if in_list[j] <
min_value:
                min_value =
in_list[j]
                min_index = j
        # swap the min w/ the
current value
        in_list[min_index] =
in_list[i]
        in_list[i] = min_value
    return None
def insertion_sort(in_list):
    """Implementation of the
Insertion Sort algorithm (in-
place)."""
    n = len(in_list)
    for i in range(n-1):
        j = i + 1
        value = in_list[j]
        while j>=1 and
value<in_list[j-1]:
            in_list[j] =
in_list[j-1]
            in_list[j-1] = value
        j -= 1

```

## Sort Algorithms (cont)

```

return None
def merge_sort(in_list):
    """Implementation of the Merge
Sort algorithm (Return the sorted
list)."""
    n = len(in_list)
    if n <= 1:
        return in_list
    right_half_sorted =
merge_sort(in_list[n/2:])
    left_half_sorted =
merge_sort(in_list[:n/2])
    return merge(left_half_sorted,
right_half_sorted)
def merge(list1, list2):
    """Merge two sorted lists such
that the merged list is also
sorted."""
    merged = []
    while len(list1)>0 and
len(list2)>0:
        if list1[0] <= list2[0]:
            merged.append(list1.pop
(0))
        else:
            merged.append(list2.pop
(0))
    merged.extend(list1)
    merged += list2
    return merged

```