## Problem 0 Source

```
 1  class Solution:
 2      def is_palindrome(s):
 3          i = 0
 4          j = len(s) - 1
 5          while i < j:
 6              if s[i] != s[j]:
 7                  return False
 8              i += 1
 9              j -= 1
10
11          return True
12
13  s = Solution()
14  print s.is_palindrome("foobar")
15  print s.is_palindrome("hannah")
```

## Problem 0 Output

```
Traceback (most recent call last):
  File "Problem 0", line 14, in <module>
    print s.is_palindrome("foobar")
TypeError: is_palindrome() takes exactly 1 argument (2 given)
```

## Problem 1 Source

```
 1  class Solution:
 2      def is_palindrome(self, s):
 3          i = 0
 4          j = len(s) - 1
 5          while i < j:
 6              if s[i] != s[j]:
 7                  return False
 8              i += 1
 9              j -= 1
10
11          return True
12
13  print Solution.is_palindrome("foobar")
14  print Solution.is_palindrome("hannah")
```

## Problem 1 Output

```
Traceback (most recent call last):
  File "Problem 1", line 13, in <module>
    print Solution.is_palindrome("foobar")
TypeError: unbound method is_palindrome() must be called with
Solution instance as first argument (got str instance instead)
```

## Problem 2 Source

```python
# counts the number of occurrences of character `c` in the string s
def count_character(s, c):
    count = 0
    for char in s:
        if char = c:
            count += 1

    return count
```

## Problem 2 Output

```
  File "Problem 2", line 5
    if char = c:
            ^
SyntaxError: invalid syntax
```

## Problem 3 Source

```python
def find_max(arr):
    maxval = arr[0]
    for a in arr:
        maxval == max(maxval, a)

    return maxval


print find_max([1,3,5,10])
```

## Problem 3 Output

```
1
```

## Problem 4 Source

```python
# looks for sub inside str
def find_substring(str, sub):
    return str.find(sub)

print find_substring("finding fens")
```

## Problem 4 Output

```
Traceback (most recent call last):
  File "Problem 4", line 5, in <module>
    print find_substring("finding fens")
TypeError: find_substring() takes exactly 2 arguments (1 given)
```

## Problem 5 Source

```python
def is_palindrome(s, i, j):
    if s[i] != s[j]:
        return False

    return is_palindrome(s, i+1, j-1)

print is_palindrome("foobar", 0, len("foobar") - 1)
print is_palindrome("hannah", 0, len("hannah") - 1)
```

## Problem 5 Output

```
False
Traceback (most recent call last):
  File "Problem 5", line 8, in <module>
    print is_palindrome("hannah", 0, len("hannah") - 1)
  File "Problem 5", line 5, in is_palindrome
    return is_palindrome(s, i+1, j-1)
  File "Problem 5", line 5, in is_palindrome
    return is_palindrome(s, i+1, j-1)
  File "Problem 5", line 5, in is_palindrome
    return is_palindrome(s, i+1, j-1)
  File "Problem 5", line 5, in is_palindrome
    return is_palindrome(s, i+1, j-1)
  File "Problem 5", line 5, in is_palindrome
    return is_palindrome(s, i+1, j-1)
  File "Problem 5", line 5, in is_palindrome
    return is_palindrome(s, i+1, j-1)
  File "Problem 5", line 2, in is_palindrome
    if s[i] != s[j]:
IndexError: string index out of range
```

## Problem 6 Source

```python
# generates all subsets from array
def all_subsets(arr, i=0, cur=[], output=[]):
    if i >= len(arr):
        output.append(cur)
        return

    all_subsets(arr, i+1, cur, output)

    cur.append(arr[i])
    all_subsets(arr, i+1, cur, output)
    cur.pop()
    return output

# should print the following lists
# [], [1], [2], [3], [1,2], [1,3], [2,3], [1,2,3]
# but prints [[], [], [], [], [], [], [], []] instead
print all_subsets([1,2,3])
```

## Problem 6 Output

```
[[], [], [], [], [], [], [], []]
```

## Problem 7 Source

```python
import sys
sys.setrecursionlimit(5)

def sum(arr, i=0):
    return sum(arr, i+1) + arr[i]

print sum([3, 5, 2])
```

## Problem 7 Output

```
Traceback (most recent call last):
  File "Problem 7", line 7, in <module>
    print sum([3, 5, 2])
  File "Problem 7", line 5, in sum
    return sum(arr, i+1) + arr[i]
  File "Problem 7", line 5, in sum
    return sum(arr, i+1) + arr[i]
  File "Problem 7", line 5, in sum
    return sum(arr, i+1) + arr[i]
  File "Problem 7", line 5, in sum
    return sum(arr, i+1) + arr[i]
RuntimeError: maximum recursion depth exceeded
```

## Problem 8 Source

```
 1  # list all duplicate values in arr
 2  def find_duplicates(arr):
 3      dupes = set()
 4      seen = {}
 5      for a in arr:
 6          if seen[a] == True:
 7              dupes.add(a)
 8          else:
 9              seen[a] = True
10
11      return seen
12
13  print find_duplicates([1,2,3,2,4,5,1])
```

## Problem 8 Output

```
Traceback (most recent call last):
  File "Problem 8", line 13, in <module>
    print find_duplicates([1,2,3,2,4,5,1])
  File "Problem 8", line 6, in find_duplicates
    if seen[a] == True:
KeyError: 1
```

## Problem 9 Source

```
def prod(arr, i=0):
    if i >= len(arr):
        return 1

    return arr[i] * prod(arr, i+1)

# if you run this file nothing happens, why?
```

## Problem 9 Output

## Problem 10 Source

```
def sum(arr):
    s = 0
    for i in xrange(arr):
        s += arr[i]
    return s

arr = [1,3,5,7,9]
print sum(arr)
```

## Problem 10 Output

```
Traceback (most recent call last):
  File "Problem 10", line 8, in <module>
    print sum(arr)
  File "Problem 10", line 3, in sum
    for i in xrange(arr):
TypeError: an integer is required
```

## Problem 11 Source

```
def is_palindrome(s, i, j):
    if i >= j:
        return True

    if s[i] != s[j]:
        return False

    return is_palindrome(s, i+1, j-1)

print is_palindrome("foobar")
print is_palindrome("hannah")

```

## Problem 11 Output

```
Traceback (most recent call last):
  File "Problem 11", line 10, in <module>
    print is_palindrome("foobar")
TypeError: is_palindrome() takes exactly 3 arguments (1 given)
```

## Problem 12 Source

```
def find_max(arr):
    maxval = arr[0]
    for val in arr:
        maxval = max(val, maxval)


print find_max([10, 3, 9, 8, 21])
```

## Problem 12 Output

```
None
```

## Problem 13 Source

```python
def sum(arr, i=0):
    if i > len(arr):
        return 0

    return sum(arr, i+1) + arr[i]

print sum([1,2,3])
```

## Problem 13 Output

```
Traceback (most recent call last):
  File "Problem 13", line 7, in <module>
    print sum([1,2,3])
  File "Problem 13", line 5, in sum
    return sum(arr, i+1) + arr[i]
  File "Problem 13", line 5, in sum
    return sum(arr, i+1) + arr[i]
  File "Problem 13", line 5, in sum
    return sum(arr, i+1) + arr[i]
  File "Problem 13", line 5, in sum
    return sum(arr, i+1) + arr[i]
IndexError: list index out of range
```

## Problem 14 Source

```python
def sum(arr):
    s = 0
    i = 0
    while i <= len(arr):
        s += arr[i]
        i += 1

    return s


print sum([1,2,3,4,5])
```

## Problem 14 Output

```
Traceback (most recent call last):
  File "Problem 14", line 11, in <module>
    print sum([1,2,3,4,5])
  File "Problem 14", line 5, in sum
    s += arr[i]
IndexError: list index out of range
```

## Problem 15 Source

```python
# calculates whether n is prime or not
def is_prime(n):
    ret = True
    for i in xrange(1, n):
        if n % i == 0:
            ret = False
        else:
            ret = True

    return ret

print is_prime(10)
print is_prime(24)
print is_prime(37)
print is_prime(43)
```

## Problem 15 Output

```
True
True
True
True
```

## Problem 16 Source

```python
# Implement the function strStr().  strStr takes two parameters a main string
# (haystack) and a substring (needle) and returns the the first index of the
# match. If there is no match, the function will return -1
# i.e if haystack = "foo bar bar" and needle = "bar" the function will return 4

# if the needle is an empty string, the haystack is returned


def strStr(haystack, needle):
    if len(needle) == 0:
        return haystack

    for i in range(len(haystack) - len(needle)):
        if haystack[i: len(needle)] == needle:
            return i
    return -1

print(strStr("endless need for needles", "needle")) # the function should return 17
```

## Problem 16 Output

```
-1
```

## Problem 17 Source

```
 1  # this takes in a string like "abc" and
 2  # generates all strings created by inserting
 3  # a space at every position in the string.
 4  # in this case, all strings will be
 5  # "a bc" and "ab c"
 6  # for "abcd", all strings will be
 7  # "a bcd", "ab cd", "abc d"
 8  def generate_all_words(s):
 9      ret = []
10      for i in xrange(1, len(s)-1):
11          c = str(s)
12          c[i] = " "
13          ret.append(c)
14
15      return ret
16
17  generate_all_words("abcd")
```

## Problem 17 Output

```
Traceback (most recent call last):
  File "Problem 17", line 17, in <module>
    generate_all_words("abcd")
  File "Problem 17", line 12, in generate_all_words
    c[i] = " "
TypeError: 'str' object does not support item assignment
```

## Problem 18 Source

```
 1  def is_palindrome(s):
 2      i = 0
 3      j = len(s) - 1
 4      while i < j:
 5          if s[i] != s[j]:
 6              return false
 7          i += 1
 8          j -= 1
 9
10      return true
11
12  is_palindrome("foobar")
13  is_palindrome("hannah")
```

## Problem 18 Output

```
Traceback (most recent call last):
  File "Problem 18", line 12, in <module>
    is_palindrome("foobar")
  File "Problem 18", line 6, in is_palindrome
    return false
NameError: global name 'false' is not defined
```

## Problem 19 Source

```
# returns if string s has the string 'needle' in it
# i.e. has_needle("has needle") should return True
# has_needle("foobar") should return False
def has_needle(s):
    return s.find(needle) != -1


print has_needle("foobar")
print has_needle("has a needle")
```

## Problem 19 Output

```
Traceback (most recent call last):
  File "Problem 19", line 8, in <module>
    print has_needle("foobar")
  File "Problem 19", line 5, in has_needle
    return s.find(needle) != -1
NameError: global name 'needle' is not defined
```