**Task2**

The parameter weekday is True if it is a weekday, and the parameter vacation is True if we are on vacation. We sleep in if it is not a weekday or we're on vacation. Return True if we sleep in.

def sleep\_in(weekday, vacation):

if not weekday or vacation:

return True

else:

return False

# This can be shortened to: return(not weekday or vacation)

**Task3**

We have two monkeys, a and b, and the parameters a\_smile and b\_smile indicate if each is smiling. We are in trouble if they are both smiling or if neither of them is smiling. Return True if we are in trouble.

def monkey\_trouble(a\_smile, b\_smile):

if a\_smile and b\_smile:

return True

if not a\_smile and not b\_smile:

return True

return False

## The above can be shortened to:

## return ((a\_smile and b\_smile) or (not a\_smile and not b\_smile))

## Or this very short version (think about how this is the same as the above)

## return (a\_smile == b\_smile)

**Task4**

Given two int values, return their sum. Unless the two values are the same, then return double their sum.

def sum\_double(a, b):

# Store the sum in a local variable

sum = a + b

# Double it if a and b are the same

if a == b:

sum = sum \* 2

return sum

**Task 5**

Given an int n, return the absolute difference between n and 21, except return double the absolute difference if n is over 21.

def diff21(n):

if n <= 21:

return 21 - n

else:

return (n - 21) \* 2

**Task6**

We have a loud talking parrot. The "hour" parameter is the current hour time in the range 0..23. We are in trouble if the parrot is talking and the hour is before 7 or after 20. Return True if we are in trouble.

def parrot\_trouble(talking, hour):

return (talking and (hour < 7 or hour > 20))

# Need extra parenthesis around the or clause

# since and binds more tightly than or.

# and is like arithmetic \*, or is like arithmetic +

**Task7**

Given 2 ints, a and b, return True if one if them is 10 or if their sum is 10.

def makes10(a, b):

return (a == 10 or b == 10 or a+b == 10)

**Task8**

Given 2 int values, return True if one is negative and one is positive. Except if the parameter "negative" is True, then return True only if both are negative.

def pos\_neg(a, b, negative):

if negative:

return (a < 0 and b < 0)

else:

return ((a < 0 and b > 0) or (a > 0 and b < 0))

**Task9**

Given a string, return a new string where "not " has been added to the front. However, if the string already begins with "not", return the string unchanged.

def not\_string(str):

if len(str) >= 3 and str[:3] == "not":

return str

return "not " + str

# str[:3] goes from the start of the string up to but not

# including index 3

**Task 10**

Given a string and a non-negative int n, return a larger string that is n copies of the original string.

def string\_times(str, n):

result = ""

for i in range(n): # range(n) is [0, 1, 2, .... n-1]

result = result + str # could use += here

return result

**Task 11**

Given a string and a non-negative int n, we'll say that the front of the string is the first 3 chars, or whatever is there if the string is less than length 3. Return n copies of the front;

def front\_times(str, n):

front\_len = 3

if front\_len > len(str):

front\_len = len(str)

front = str[:front\_len]

result = ""

for i in range(n):

result = result + front

return result

**Task12**

Given a string, return a new string made of every other char starting with the first, so "Hello" yields "Hlo".

def string\_bits(str):

result = ""

# Many ways to do this. This uses the standard loop of i on every char,

# and inside the loop skips the odd index values.

for i in range(len(str)):

if i % 2 == 0:

result = result + str[i]

return result

**Task13**

Given a non-empty string like "Code" return a string like "CCoCodCode".

def string\_splosion(str):

result = ""

# On each iteration, add the substring of the chars 0..i

for i in range(len(str)):

result = result + str[:i+1]

return result