**Completed WarmUp 1 and WarmUp 2 problems from** [**Codingbat**](https://codingbat.com/python)

**WARMUP 1**

1. **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.**

**sleep\_in(False, False) → True**

**sleep\_in(True, False) → False**

**sleep\_in(False, True) → True**

**def sleep\_in(weekday, vacation):**

**if not weekday and not vacation:**

**return True**

**if weekday and not vacation:**

**return False**

**if not weekday and vacation:**

**return True**

**if weekday and vacation:**

**return True**

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

**diff21(19) → 2**

**diff21(10) → 11**

**diff21(21) → 0**

**def diff21(n):**

**if n>21:**

**return 2\*abs(21-n)**

**else:**

**return abs(21-n)**

**3)Given an int n, return True if it is within 10 of 100 or 200. Note: abs(num) computes the absolute value of a number.**

**near\_hundred(93) → True**

**near\_hundred(90) → True**

**near\_hundred(89) → False**

**def near\_hundred(n):**

**if (n<=110 and n>=90) or (n<=210 and n>=190):**

**return True**

**else:**

**return False**

**4)Given a non-empty string and an int n, return a new string where the char at index n has been removed. The value of n will be a valid index of a char in the original string (i.e. n will be in the range 0..len(str)-1 inclusive).**

**missing\_char('kitten', 1) → 'ktten'**

**missing\_char('kitten', 0) → 'itten'**

**missing\_char('kitten', 4) → 'kittn'**

**def missing\_char(str, n):**

**i=str[:n]+''+str[n+1:]**

**return i**

**5)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.**

**monkey\_trouble(True, True) → True**

**monkey\_trouble(False, False) → True**

**monkey\_trouble(True, False) → False**

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

**if (a\_smile and b\_smile) or (not a\_smile and not b\_smile):**

**return True**

**else:**

**return False**

**6)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.**

**parrot\_trouble(True, 6) → True**

**parrot\_trouble(True, 7) → False**

**parrot\_trouble(False, 6) → False**

**def parrot\_trouble(talking, hour):**

**if (talking and hour<7) or (talking and hour>20):**

**return True**

**else:**

**return False**

**7) 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.**

**pos\_neg(1, -1, False) → True**

**pos\_neg(-1, 1, False) → True**

**pos\_neg(-4, -5, True) → True**

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

**if negative:**

**if a<0 and b<0:**

**return True**

**else:**

**return False**

**elif (a<0 and b>0) or (a>0 and b<0):**

**return True**

**else:**

**return False**

**8) Given a string, return a new string where the first and last chars have been exchanged.**

**front\_back('code') → 'eodc'**

**front\_back('a') → 'a'**

**front\_back('ab') → 'ba'**

**def front\_back(str):**

**if len(str) <= 1:**

**return str**

**else:**

**mid = str[1:len(str)-1] # can be written as str[1:-1]**

**return str[len(str)-1] + mid + str[0]**

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

**sum\_double(1, 2) → 3**

**sum\_double(3, 2) → 5**

**sum\_double(2, 2) → 8**

**def sum\_double(a, b):**

**if a==b:**

**return 2\*(a+b)**

**else:**

**return a+b**

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

**makes10(9, 10) → True**

**makes10(9, 9) → False**

**makes10(1, 9) → True**

**def makes10(a, b):**

**if a==10 or b==10 or a+b==10:**

**return True**

**else:**

**return False**

|  |
| --- |
| **11) 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.**  **not\_string('candy') → 'not candy'**  **not\_string('x') → 'not x'**  **not\_string('not bad') → 'not bad'**  **def not\_string(str):**  **if str[:3]=='not':**  **return str**  **else:**  **return 'not'+' '+str** |
| **12) Given a string, we'll say that the front is the first 3 chars of the string. If the string length is less than 3, the front is whatever is there. Return a new string which is 3 copies of the front.**  **front3('Java') → 'JavJavJav'**  **front3('Chocolate') → 'ChoChoCho'**  **front3('abc') → 'abcabcabc'** |

**def front3(str):**

**if len(str)<3:**

**return str+str+str**

**else:**

**return str[:3]+str[:3]+str[:3]**

**WARMUP 2**

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

**string\_times('Hi', 2) → 'HiHi'**

**string\_times('Hi', 3) → 'HiHiHi'**

**string\_times('Hi', 1) → 'Hi'**

**def string\_times(str, n):**

**i=0**

**result=""**

**while i<n:**

**result=result+str**

**i+=1**

**return result**

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

**string\_splosion('Code') → 'CCoCodCode'**

**string\_splosion('abc') → 'aababc'**

**string\_splosion('ab') → 'aab'**

**def string\_splosion(str):**

**i=0**

**result=''**

**while i<=len(str):**

**result+=str[:i]**

**i+=1**

**return result**

**3) Given an array of ints, return True if one of the first 4 elements in the array is a 9. The array length may be less than 4.**

**array\_front9([1, 2, 9, 3, 4]) → True**

**array\_front9([1, 2, 3, 4, 9]) → False**

**array\_front9([1, 2, 3, 4, 5]) → False**

**def array\_front9(nums):**

**l=len(nums)**

**i=0**

**if l>4:**

**l=4**

**while i<l:**

**if nums[i]==9:**

**return True**

**i+=1**

**return False**

**4) 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;**

**front\_times('Chocolate', 2) → 'ChoCho'**

**front\_times('Chocolate', 3) → 'ChoChoCho'**

**front\_times('Abc', 3) → 'AbcAbcAbc'**

**def front\_times(str, n):**

**l=len(str)**

**i=0**

**res=''**

**front\_len=3**

**while i<n:**

**res=res+str[:front\_len]**

**i+=1**

**return res**

**5) Given a string, return the count of the number of times that a substring length 2 appears in the string and also as the last 2 chars of the string, so "hixxxhi" yields 1 (we won't count the end substring).**

**last2('hixxhi') → 1**

**last2('xaxxaxaxx') → 1**

**last2('axxxaaxx') → 2**

**def last2(str):**

**l=len(str)**

**if l<2:**

**return 0**

**last2=str[l-2:]**

**if last2:**

**count = 0**

**for i in range(l-2):**

**sub=str[i:i+2]**

**if sub==last2:**

**count = count+1**

**return count**

**6) Given an array of ints, return True if the sequence of numbers 1, 2, 3 appears in the array somewhere.**

**array123([1, 1, 2, 3, 1]) → True**

**array123([1, 1, 2, 4, 1]) → False**

**array123([1, 1, 2, 1, 2, 3]) → True**

**def array123(nums):**

**for i in range(len(nums)-2):**

**if nums[i]==1 and nums[i+1]==2 and nums[i+2]==3:**

**return True**

**return False**

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

**string\_bits('Hello') → 'Hlo'**

**string\_bits('Hi') → 'H'**

**string\_bits('Heeololeo') → 'Hello'**

**def string\_bits(str):**

**res=''**

**for i in range(len(str)):**

**if i%2==0:**

**res=res+str[i]**

**return res**

|  |
| --- |
| **8) Given an array of ints, return the number of 9's in the array.**  **array\_count9([1, 2, 9]) → 1**  **array\_count9([1, 9, 9]) → 2**  **array\_count9([1, 9, 9, 3, 9]) → 3**  **def array\_count9(nums):**  **count=0**  **l=len(nums)**  **for i in range(l):**  **if nums[i]==9:**  **count+=1**  **return count**  **9) Given 2 strings, a and b, return the number of the positions where they contain the same length 2 substring. So "xxcaazz" and "xxbaaz" yields 3, since the "xx", "aa", and "az" substrings appear in the same place in both strings.**  **string\_match('xxcaazz', 'xxbaaz') → 3**  **string\_match('abc', 'abc') → 2**  **string\_match('abc', 'axc') → 0**  **def string\_match(a, b):**  **count=0;**  **for i in range(len(a)-1):**  **sub1=a[i:i+2]**  **sub2=b[i:i+2]**  **if sub1==sub2:**  **count=count+1**  **return count** |
|  |