

KASTAMONU ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

PROJE RAPORU

Programlama Dilleri

Coding Bat Çözümleri

184410029

RECEP POLAT

Bu raporda 72 adet soru ve 8 konu vardır. Konular ve Sorular:

* Warmup-1 (12 Soru)
* sleep\_in, [monkey\_trouble](https://codingbat.com/prob/p120546), sum\_double, diff21, parrot\_trouble, makes10, near\_hundred, pos\_neg, not\_string, missing\_char, front\_back, front3
* Warmup-2 (9 Soru)
* string\_times, front\_times, string\_bits, string\_splosion, last2, array\_count9, array\_front9, array123, string\_match
* Logic-1 (9 Soru)
* cigar\_party, date\_fashion, squirrel\_play, caught\_speeding, alarm\_clock, sorta\_sum, love5, in1to10, near\_ten
* Logic-2 (7 Soru)
* make\_bricks, lone\_sum, lucky\_sum, no\_teen\_sum, round\_sum, close\_far, make\_chocolate
* String-1 (11 Soru)
* Hello\_name, make\_abba, make\_tags, make\_out\_word, extra\_end, first\_two, first\_half, without\_end, combo\_string, non\_start, left2
* String-2 (6 Soru)
* Double\_char, count\_hi, cat\_dog, count\_code, end\_other, xyz\_there
* List-1 (12 Soru)
* first\_last6, same\_first\_last, make\_pi, common\_end, sum3, rotate\_left3, reverse3, max\_end3, sum2, middle\_way, make\_ends, has23
* List-2 (6 Soru)
* Count\_evens, big\_diff, centered\_average, sum13, sum67, has22

Renkler

String Değeri

Hazır Fonksiyon

Int ve Bool Değeri

Warmup-1

1. **sleep\_in**

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):  return (weekday == False and vacation == False) or vacation == True |

1. **monkey\_trouble**

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):  return (a\_smile == True and b\_smile == True) or (a\_smile == False and b\_smile == False) |

1. **sum\_double**

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 a + b  else:  return 2 \* (a + b) |

1. **diff21**

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 abs(21 - n)  else:  return 2 \* (n - 21) |

1. **parrot\_trouble**

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):  return talking == True and (hour < 7 or hour > 20 ) |

1. **makes10**

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):  return a == 10 or b == 10 or a + b == 10 |

1. **near\_hundred**

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):  return abs(100 - n) <= 10 or abs(200 - n) <= 10 |

1. **pos\_neg**

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):  return (((a < 0 and b > 0) or (a > 0 and b < 0)) and negative == False) or (a < 0 and b < 0 and negative == True) |

1. **not\_string**

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 "not" == str[0:3]:  return str  else:  return "not " + str |

1. **missing\_char**

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):  return str[0:n] + str[n+1:] |

1. **front\_back**

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  return str[len(str)-1] + str[1:len(str) - 1] + str[0] |

1. **front3**

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):  return str[0:3] + str[0:3]+ str[0:3] |

Warmup-2

1. **string\_times**

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):  txt = ""  for i in range(n):  txt += str  return txt |

1. **front\_times**

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):  txt = ""  for i in range(n):  txt += str[0:3]  return txt |

1. **string\_bits**

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):  txt = ""  for i in range(len(str)):  if i % 2 == 0:  txt += str[i]  return txt |

1. **string\_splosion**

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):  txt = ""  for i in range(len(str)):  txt += str[0:i+1]  return txt |

1. **last2**

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):  if len(str) < 2:  return 0  sayac = 0  for i in range(len(str) - 2):  if str[i:i + 2] == str[len(str) - 2:]:  sayac += 1  return sayac |

1. **array\_count9**

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):  sayac = 0  for i in nums:  if i == 9:  sayac += 1  return sayac |

1. **array\_front9**

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):  for i in range(len(nums)):  if i <= 3 and nums[i] == 9:  return True  return False |

1. **array123**

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 |

1. **string\_match**

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):  sayac = 0  minik = min(len(a),len(b))  for i in range(minik-1):  if a[i:i+2] == b[i:i+2]:  sayac += 1  return sayac |

Logic-1

1. **cigar\_party**

When squirrels get together for a party, they like to have cigars. A squirrel party is successful when the number of cigars is between 40 and 60, inclusive. Unless it is the weekend, in which case there is no upper bound on the number of cigars. Return True if the party with the given values is successful, or False otherwise.

cigar\_party(30, False) → False  
cigar\_party(50, False) → True  
cigar\_party(70, True) → True

|  |
| --- |
| def cigar\_party(cigars, is\_weekend):  return (cigars <= 60 and cigars >= 40 and is\_weekend == False) or (is\_weekend == True and cigars >= 40) |

1. **date\_fashion**

You and your date are trying to get a table at a restaurant. The parameter "you" is the stylishness of your clothes, in the range 0..10, and "date" is the stylishness of your date's clothes. The result getting the table is encoded as an int value with 0=no, 1=maybe, 2=yes. If either of you is very stylish, 8 or more, then the result is 2 (yes). With the exception that if either of you has style of 2 or less, then the result is 0 (no). Otherwise the result is 1 (maybe).

date\_fashion(5, 10) → 2  
date\_fashion(5, 2) → 0  
date\_fashion(5, 5) → 1

|  |
| --- |
| def date\_fashion(you, date):  if (you <= 2) or (date <= 2):  return 0  elif (you >= 8) or (date >= 8):  return 2  else:  return 1 |

1. **squirrel\_play**

The squirrels in Palo Alto spend most of the day playing. In particular, they play if the temperature is between 60 and 90 (inclusive). Unless it is summer, then the upper limit is 100 instead of 90. Given an int temperature and a boolean is\_summer, return True if the squirrels play and False otherwise.

squirrel\_play(70, False) → True  
squirrel\_play(95, False) → False  
squirrel\_play(95, True) → True

|  |
| --- |
| def squirrel\_play(temp, is\_summer):  if is\_summer:  return temp >= 60 and temp <= 100  else:  return temp >= 60 and temp <= 90 |

1. **caught\_speeding**

You are driving a little too fast, and a police officer stops you. Write code to compute the result, encoded as an int value: 0=no ticket, 1=small ticket, 2=big ticket. If speed is 60 or less, the result is 0. If speed is between 61 and 80 inclusive, the result is 1. If speed is 81 or more, the result is 2. Unless it is your birthday -- on that day, your speed can be 5 higher in all cases.

caught\_speeding(60, False) → 0  
caught\_speeding(65, False) → 1  
caught\_speeding(65, True) → 0

|  |
| --- |
| def caught\_speeding(speed, is\_birthday):  bilet = -1  hiz = 0  if is\_birthday:  hiz = 5  else:  hiz = 0  if speed <= 60 + hiz :  bilet = 0  elif speed <= 80 + hiz:  bilet = 1  elif speed > 80:  bilet = 2  return bilet |

1. **alarm\_clock**

Given a day of the week encoded as 0=Sun, 1=Mon, 2=Tue, ...6=Sat, and a boolean indicating if we are on vacation, return a string of the form "7:00" indicating when the alarm clock should ring. Weekdays, the alarm should be "7:00" and on the weekend it should be "10:00". Unless we are on vacation -- then on weekdays it should be "10:00" and weekends it should be "off".

alarm\_clock(1, False) → '7:00'  
alarm\_clock(5, False) → '7:00'  
alarm\_clock(0, False) → '10:00'

|  |
| --- |
| def alarm\_clock(day, vacation):  if vacation:  if day >= 1 and day <= 5:  return “10:00”  else:  return “off'”  else:  if day >= 1 and day <= 5:  return ”7:00”  return “10:00” |

1. **sorta\_sum**

Given 2 ints, a and b, return their sum. However, sums in the range 10..19 inclusive, are forbidden, so in that case just return 20.

sorta\_sum(3, 4) → 7  
sorta\_sum(9, 4) → 20  
sorta\_sum(10, 11) → 21

|  |
| --- |
| def sorta\_sum(a, b):  if a + b >= 10 and a + b <= 19:  return 20  return a + b |

1. **love6**

The number 6 is a truly great number. Given two int values, a and b, return True if either one is 6. Or if their sum or difference is 6. Note: the function abs(num) computes the absolute value of a number.

love6(6, 4) → True  
love6(4, 5) → False  
love6(1, 5) → True

|  |
| --- |
| def love6(a, b):  return (a == 6 or b == 6) or (a + b == 6) or abs(a - b) == 6 |

1. **in1to10**

Given a number n, return True if n is in the range 1..10, inclusive. Unless outside\_mode is True, in which case return True if the number is less or equal to 1, or greater or equal to 10.

in1to10(5, False) → True  
in1to10(11, False) → False  
in1to10(11, True) → True

|  |
| --- |
| def in1to10(n, outside\_mode):  if outside\_mode:  if n < 10 and n > 1:  return False  return True  else:  if n <= 10 and n >= 1:  return True  return False |

1. **near\_ten**

Given a non-negative number "num", return True if num is within 2 of a multiple of 10. Note: (a % b) is the remainder of dividing a by b, so (7 % 5) is 2. See also: [Introduction to Mod](https://codingbat.com/doc/practice/mod-introduction.html)

near\_ten(12) → True  
near\_ten(17) → False  
near\_ten(19) → True

|  |
| --- |
| def near\_ten(num):  if num % 10 <= 2:  return True  else:  if 10 - num % 10 <= 2:  return True  return False |

Logic-2

1. **make\_bricks**

We want to make a row of bricks that is **goal** inches long. We have a number of small bricks (1 inch each) and big bricks (5 inches each). Return True if it is possible to make the goal by choosing from the given bricks. This is a little harder than it looks and can be done without any loops. See also: [Introduction to MakeBricks](https://codingbat.com/doc/practice/makebricks-introduction.html)

make\_bricks(3, 1, 8) → True  
make\_bricks(3, 1, 9) → False  
make\_bricks(3, 2, 10) → True

|  |
| --- |
| def make\_bricks(small, big, goal):  if goal > small + big \* 5:  return False  return goal % 5 <= small |

1. **lone\_sum**

Given 3 int values, a b c, return their sum. However, if one of the values is the same as another of the values, it does not count towards the sum.

lone\_sum(1, 2, 3) → 6  
lone\_sum(3, 2, 3) → 2  
lone\_sum(3, 3, 3) → 0

|  |
| --- |
| def lone\_sum(a, b, c):  if a == b == c:  return 0  toplam = 0  if a != b and a != c:  toplam+= a  if b != a and b != c:  toplam += b  if c != a and c != b:  toplam += c  return sum |

1. **lucky\_sum**

Given 3 int values, a b c, return their sum. However, if one of the values is 13 then it does not count towards the sum and values to its right do not count. So for example, if b is 13, then both b and c do not count.

lucky\_sum(1, 2, 3) → 6  
lucky\_sum(1, 2, 13) → 3  
lucky\_sum(1, 13, 3) → 1

|  |
| --- |
| def lucky\_sum(a, b, c):  if a == 13:  return 0  elif b == 13:  return a  elif c == 13:  return a + b  return a + b + c |

1. **no\_teen\_sum**

Given 3 int values, a b c, return their sum. However, if any of the values is a teen -- in the range 13..19 inclusive -- then that value counts as 0, except 15 and 16 do not count as a teens. Write a separate helper "def fix\_teen(n):"that takes in an int value and returns that value fixed for the teen rule. In this way, you avoid repeating the teen code 3 times (i.e. "decomposition"). Define the helper below and at the same indent level as the main no\_teen\_sum().

no\_teen\_sum(1, 2, 3) → 6  
no\_teen\_sum(2, 13, 1) → 3  
no\_teen\_sum(2, 1, 14) → 3

|  |
| --- |
| def no\_teen\_sum(a, b, c):  return fix\_teen(a) + fix\_teen(b) + fix\_teen(c)  def fix\_teen(n):  if n in [13, 14, 17, 18, 19]:  return 0  return n |

1. **round\_sum**

For this problem, we'll round an int value up to the next multiple of 10 if its rightmost digit is 5 or more, so 15 rounds up to 20. Alternately, round down to the previous multiple of 10 if its rightmost digit is less than 5, so 12 rounds down to 10. Given 3 ints, a b c, return the sum of their rounded values. To avoid code repetition, write a separate helper "def round10(num):" and call it 3 times. Write the helper entirely below and at the same indent level as round\_sum().

round\_sum(16, 17, 18) → 60  
round\_sum(12, 13, 14) → 30  
round\_sum(6, 4, 4) → 10

|  |
| --- |
| def round\_sum(a, b, c):  return round10(a) + round10(b) + round10(c)  def round10(num):  if num % 10 >= 5:  return (int)(num / 10) \* 10 + 10  else:  return (int)(num / 10) \* 10 |

1. **close\_far**

Given three ints, a b c, return True if one of b or c is "close" (differing from a by at most 1), while the other is "far", differing from both other values by 2 or more. Note: abs(num) computes the absolute value of a number.

close\_far(1, 2, 10) → True  
close\_far(1, 2, 3) → False  
close\_far(4, 1, 3) → True

|  |
| --- |
| def close\_far(a, b, c):  if abs(b-a) <= 1:  close = b  elif abs(c-a) <= 1:  close = c  else:  return False  if (close == b) and (abs(c-a) >= 2) and (abs(c-b)>=2):  return True  elif (close == c) and (abs(b-a) >= 2) and (abs(b-c)>=2):  return True  else:  return False |

1. **make\_chocolate**

We want make a package of **goal** kilos of chocolate. We have small bars (1 kilo each) and big bars (5 kilos each). Return the number of small bars to use, assuming we always use big bars before small bars. Return -1 if it can't be done.

make\_chocolate(4, 1, 9) → 4  
make\_chocolate(4, 1, 10) → -1  
make\_chocolate(4, 1, 7) → 2

|  |
| --- |
| def make\_chocolate(small, big, goal):  maxBig = goal / 5  if maxBig >= big:  if goal <= big \* 5 + small:  return goal – big \* 5  if big > maxBig:  if goal <= maxBig \* 5 +small:  return goal - maxBig \* 5  return -1 |

String-1

1. **hello\_name**

Given a string name, e.g. "Bob", return a greeting of the form "Hello Bob!".

hello\_name('Bob') → 'Hello Bob!'  
hello\_name('Alice') → 'Hello Alice!'  
hello\_name('X') → 'Hello X!'

|  |
| --- |
| def hello\_name(name):  return("Hello " + name + "!") |

1. **make\_abba**

Given two strings, a and b, return the result of putting them together in the order abba, e.g. "Hi" and "Bye" returns "HiByeByeHi".

make\_abba('Hi', 'Bye') → 'HiByeByeHi'  
make\_abba('Yo', 'Alice') → 'YoAliceAliceYo'  
make\_abba('What', 'Up') → 'WhatUpUpWhat'

|  |
| --- |
| def make\_abba(a, b):  return(a + b + b + a) |

1. **make\_tags**

The web is built with HTML strings like "<i>Yay</i>" which draws Yay as italic text. In this example, the "i" tag makes <i> and </i> which surround the word "Yay". Given tag and word strings, create the HTML string with tags around the word, e.g. "<i>Yay</i>".

make\_tags('i', 'Yay') → '<i>Yay</i>'  
make\_tags('i', 'Hello') → '<i>Hello</i>'  
make\_tags('cite', 'Yay') → '<cite>Yay</cite>'

|  |
| --- |
| def make\_tags(tag, word):  return("<" + tag + ">" + word + "</" + tag + ">") |

1. **make\_out\_word**

Given an "out" string length 4, such as "<<>>", and a word, return a new string where the word is in the middle of the out string, e.g. "<<word>>".

make\_out\_word('<<>>', 'Yay') → '<<Yay>>'  
make\_out\_word('<<>>', 'WooHoo') → '<<WooHoo>>'  
make\_out\_word('[[]]', 'word') → '[[word]]'

|  |
| --- |
| def make\_out\_word(out, word):  return(out[0:2] + word + out[2:4]) |

1. **extra\_end**

Given a string, return a new string made of 3 copies of the last 2 chars of the original string. The string length will be at least 2.

extra\_end('Hello') → 'lololo'  
extra\_end('ab') → 'ababab'  
extra\_end('Hi') → 'HiHiHi'

|  |
| --- |
| def extra\_end(str):  text = ""  for i in range(3):  text += str[len(str) - 2] + str[len(str) - 1]  return(text) |

1. **first\_two**

Given a string, return the string made of its first two chars, so the String "Hello" yields "He". If the string is shorter than length 2, return whatever there is, so "X" yields "X", and the empty string "" yields the empty string ""

first\_two('Hello') → 'He'  
first\_two('abcdefg') → 'ab'  
first\_two('ab') → 'ab'

|  |
| --- |
| def first\_two(str):  return(str[0:2]) |

1. **first\_half**

Given a string of even length, return the first half. So the string "WooHoo" yields "Woo".

first\_half('WooHoo') → 'Woo'  
first\_half('HelloThere') → 'Hello'  
first\_half('abcdef') → 'abc'

|  |
| --- |
| def first\_half(str):  return(str[0:len(str) / 2]) |

1. **without\_end**

Given a string, return a version without the first and last char, so "Hello" yields "ell". The string length will be at least 2.

without\_end('Hello') → 'ell'  
without\_end('java') → 'av'  
without\_end('coding') → 'odin'

|  |
| --- |
| def without\_end(str):  return(str[1:len(str) - 1]) |

1. **combo\_string**

Given 2 strings, a and b, return a string of the form short+long+short, with the shorter string on the outside and the longer string on the inside. The strings will not be the same length, but they may be empty (length 0).

combo\_string('Hello', 'hi') → 'hiHellohi'  
combo\_string('hi', 'Hello') → 'hiHellohi'  
combo\_string('aaa', 'b') → 'baaab'

|  |
| --- |
| def combo\_string(a, b):  if len(a) > len(b):  return(b + a + b)  else:  return(a + b + a) |

1. **non\_start**

Given 2 strings, return their concatenation, except omit the first char of each. The strings will be at least length 1.

non\_start('Hello', 'There') → 'ellohere'  
non\_start('java', 'code') → 'avaode'  
non\_start('shotl', 'java') → 'hotlava'

|  |
| --- |
| def non\_start(a, b):  return(a[1:len(a)] + b[1:len(b)]) |

1. **left2**

Given a string, return a "rotated left 2" version where the first 2 chars are moved to the end. The string length will be at least 2.

left2('Hello') → 'lloHe'  
left2('java') → 'vaja'  
left2('Hi') → 'Hi'

|  |
| --- |
| def left2(str):  return(str[2:len(str)] + str[0:2]) |

String-2

1. **double\_char**

Given a string, return a string where for every char in the original, there are two chars.

double\_char('The') → 'TThhee'  
double\_char('AAbb') → 'AAAAbbbb'  
double\_char('Hi-There') → 'HHii--TThheerree'

|  |
| --- |
| def double\_char(str):  text = ""  for i in range(0,len(str)):  text += str[i] + str[i]  return text |

1. **count\_hi**

Return the number of times that the string "hi" appears anywhere in the given string.

count\_hi('abc hi ho') → 1  
count\_hi('ABChi hi') → 2  
count\_hi('hihi') → 2

|  |
| --- |
| def count\_hi(str):  count = 0  for i in range(len(str) - 1):  if str[i] == 'h' and str[i + 1] == 'i':  count += 1  return count |

1. **cat\_dog**

Return True if the string "cat" and "dog" appear the same number of times in the given string.

cat\_dog('catdog') → True  
cat\_dog('catcat') → False  
cat\_dog('1cat1cadodog') → True

|  |
| --- |
| def cat\_dog(str):  sumcat = 0  sumdog = 0  for i in range(len(str)):  if str[i:i + 3] == "cat":  sumcat += 1  elif str[i:i + 3]== "dog":  sumdog += 1  if sumcat == sumdog:  return True  return False |

1. **count\_code**

Return the number of times that the string "code" appears anywhere in the given string, except we'll accept any letter for the 'd', so "cope" and "cooe" count.

count\_code('aaacodebbb') → 1  
count\_code('codexxcode') → 2  
count\_code('cozexxcope') → 2

|  |
| --- |
| def count\_code(str):  count = 0  for i in range(len(str) - 1):  if str[i: i + 2] == "co" and str[i + 3:i + 4] == "e":  count += 1  return count |

1. **end\_other**

Given two strings, return True if either of the strings appears at the very end of the other string, ignoring upper/lower case differences (in other words, the computation should not be "case sensitive"). Note: s.lower() returns the lowercase version of a string.

end\_other('Hiabc', 'abc') → True  
end\_other('AbC', 'HiaBc') → True  
end\_other('abc', 'abXabc') → True

|  |
| --- |
| def end\_other(a, b):  index = 0  if len(a) > len(b):  index = len(a) - len(b)  if a[index:].lower() == b.lower():  return True  return False  else:  index = len(b) - len(a)  if b[index: ].lower() == a.lower():  return True  return False |

1. **xyz\_there**

Return True if the given string contains an appearance of "xyz" where the xyz is not directly preceeded by a period (.). So "xxyz" counts but "x.xyz" does not.

xyz\_there('abcxyz') → True  
xyz\_there('abc.xyz') → False  
xyz\_there('xyz.abc') → True

|  |
| --- |
| def xyz\_there(str):  for i in range(len(str)):  if str[i] != '.' and str[i+1:i+4] == 'xyz':  return True  if str[0:3] == 'xyz':  return True  return False |

List-1

1. **first\_last6**

Given an array of ints, return True if 6 appears as either the first or last element in the array. The array will be length 1 or more.

first\_last6([1, 2, 6]) → True  
first\_last6([6, 1, 2, 3]) → True  
first\_last6([13, 6, 1, 2, 3]) → False

|  |
| --- |
| def first\_last6(nums):  if nums[0] == 6 or nums[len(nums)-1] == 6:  return True  else:  return False |

1. **same\_first\_last**

Given an array of ints, return True if the array is length 1 or more, and the first element and the last element are equal.

same\_first\_last([1, 2, 3]) → False  
same\_first\_last([1, 2, 3, 1]) → True  
same\_first\_last([1, 2, 1]) → True

|  |
| --- |
| def same\_first\_last(nums):  return(len(nums) >= 1 and nums[0] == nums[len(nums) -1]) |

1. **make\_pi**

Return an int array length 3 containing the first 3 digits of pi, {3, 1, 4}.

make\_pi() → [3, 1, 4]

|  |
| --- |
| def make\_pi():  list = [3,1,4]  return list |

1. **common\_end**

Given 2 arrays of ints, a and b, return True if they have the same first element or they have the same last element. Both arrays will be length 1 or more.

common\_end([1, 2, 3], [7, 3]) → True  
common\_end([1, 2, 3], [7, 3, 2]) → False  
common\_end([1, 2, 3], [1, 3]) → True

|  |
| --- |
| def common\_end(a, b):  return(len(a) >= 1 and len(b) >= 1 and a[0] == b[0] or a[-1] == b[-1]) |

1. **sum3**

Given an array of ints length 3, return the sum of all the elements.

sum3([1, 2, 3]) → 6  
sum3([5, 11, 2]) → 18  
sum3([7, 0, 0]) → 7

|  |
| --- |
| def sum3(nums):  toplam = 0  for i in range(0,len(nums),1):  toplam += nums[i]  return toplam |

1. **rotate\_left3**

Given an array of ints length 3, return an array with the elements "rotated left" so {1, 2, 3} yields {2, 3, 1}.

rotate\_left3([1, 2, 3]) → [2, 3, 1]  
rotate\_left3([5, 11, 9]) → [11, 9, 5]  
rotate\_left3([7, 0, 0]) → [0, 0, 7]

|  |
| --- |
| def rotate\_left3(nums):  return[nums[1], nums[2] , nums[0]] |

1. **reverse3**

Given an array of ints length 3, return a new array with the elements in reverse order, so {1, 2, 3} becomes {3, 2, 1}.

reverse3([1, 2, 3]) → [3, 2, 1]  
reverse3([5, 11, 9]) → [9, 11, 5]  
reverse3([7, 0, 0]) → [0, 0, 7]

|  |
| --- |
| def reverse3(nums):  return [nums[2],nums[1],nums[0]] |

1. **max\_end3**

Given an array of ints length 3, figure out which is larger, the first or last element in the array, and set all the other elements to be that value. Return the changed array.

max\_end3([1, 2, 3]) → [3, 3, 3]  
max\_end3([11, 5, 9]) → [11, 11, 11]  
max\_end3([2, 11, 3]) → [3, 3, 3]

|  |
| --- |
| def max\_end3(nums):  if nums[0] > nums[2]:  return [nums[0],nums[0],nums[0]]  else:  return[nums[2],nums[2],nums[2]] |

1. **sum2**

Given an array of ints, return the sum of the first 2 elements in the array. If the array length is less than 2, just sum up the elements that exist, returning 0 if the array is length 0.

sum2([1, 2, 3]) → 3  
sum2([1, 1]) → 2  
sum2([1, 1, 1, 1]) → 2

|  |
| --- |
| def sum2(nums):  if len(nums) >= 2:  return nums[0] + nums[1]  elif len(nums) == 1:  return nums[0]  else:  return 0  return toplam |

1. **middle\_way**

Given 2 int arrays, a and b, each length 3, return a new array length 2 containing their middle elements.

middle\_way([1, 2, 3], [4, 5, 6]) → [2, 5]  
middle\_way([7, 7, 7], [3, 8, 0]) → [7, 8]  
middle\_way([5, 2, 9], [1, 4, 5]) → [2, 4]

|  |
| --- |
| def middle\_way(a, b):  return [a[1],b[1]] |

1. **make\_ends**

Given an array of ints, return a new array length 2 containing the first and last elements from the original array. The original array will be length 1 or more.

make\_ends([1, 2, 3]) → [1, 3]  
make\_ends([1, 2, 3, 4]) → [1, 4]  
make\_ends([7, 4, 6, 2]) → [7, 2]

|  |
| --- |
| def make\_ends(nums):  return [nums[0], nums[len(nums)-1]] |

1. **has23**

Given an int array length 2, return True if it contains a 2 or a 3.

has23([2, 5]) → True  
has23([4, 3]) → True  
has23([4, 5]) → False

|  |
| --- |
| def has23(nums):  if nums[0] == 2 or nums[0] == 3 or nums[1] == 2 or nums[1] == 3:  return True  else:  return False |

List-2

1. **count\_evens**

Return the number of even ints in the given array. Note: the % "mod" operator computes the remainder, e.g. 5 % 2 is 1.

count\_evens([2, 1, 2, 3, 4]) → 3  
count\_evens([2, 2, 0]) → 3  
count\_evens([1, 3, 5]) → 0

|  |
| --- |
| def count\_evens(nums):  sayac = 0  for i in range(len(nums)):  if nums[i] % 2 == 0:  sayac += 1  return sayac |

1. **big\_diff**

Given an array length 1 or more of ints, return the difference between the largest and smallest values in the array. Note: the built-in min(v1, v2) and max(v1, v2) functions return the smaller or larger of two values.

big\_diff([10, 3, 5, 6]) → 7  
big\_diff([7, 2, 10, 9]) → 8  
big\_diff([2, 10, 7, 2]) → 8

|  |
| --- |
| def big\_diff(nums):  return max(nums) - min(nums) |

1. **centered\_average**

Return the "centered" average of an array of ints, which we'll say is the mean average of the values, except ignoring the largest and smallest values in the array. If there are multiple copies of the smallest value, ignore just one copy, and likewise for the largest value. Use int division to produce the final average. You may assume that the array is length 3 or more.

centered\_average([1, 2, 3, 4, 100]) → 3  
centered\_average([1, 1, 5, 5, 10, 8, 7]) → 5  
centered\_average([-10, -4, -2, -4, -2, 0]) → -3

|  |
| --- |
| def centered\_average(nums):  small = min(nums)  big = max(nums)  sum = 0  for i in nums:  sum += i  return (sum - big- small)/(len(nums)-2) |

1. **sum13**

Return the sum of the numbers in the array, returning 0 for an empty array. Except the number 13 is very unlucky, so it does not count and numbers that come immediately after a 13 also do not count.

sum13([1, 2, 2, 1]) → 6  
sum13([1, 1]) → 2  
sum13([1, 2, 2, 1, 13]) → 6

|  |
| --- |
| def sum13(nums):  toplam = 0  i = 0  while i < len(nums):  if nums[i] == 13:  i += 1  else:  toplam += nums[i]  i += 1  return toplam |

1. **sum67**

Return the sum of the numbers in the array, except ignore sections of numbers starting with a 6 and extending to the next 7 (every 6 will be followed by at least one 7). Return 0 for no numbers.

sum67([1, 2, 2]) → 5  
sum67([1, 2, 2, 6, 99, 99, 7]) → 5  
sum67([1, 1, 6, 7, 2]) → 4

|  |
| --- |
| def sum67(nums):  toplam = 0  i = 0  while i < len(nums):  if nums[i] == 6:  while nums[i] != 7:  i += 1  else:  toplam += nums[i]  i += 1  return toplam |

1. **has22**

Given an array of ints, return True if the array contains a 2 next to a 2 somewhere.

has22([1, 2, 2]) → True  
has22([1, 2, 1, 2]) → False  
has22([2, 1, 2]) → False

|  |
| --- |
| def has22(nums):  for i in range(len(nums) - 1):  if nums[i] == 2 and nums[i + 1] == 2:  return True  return False |