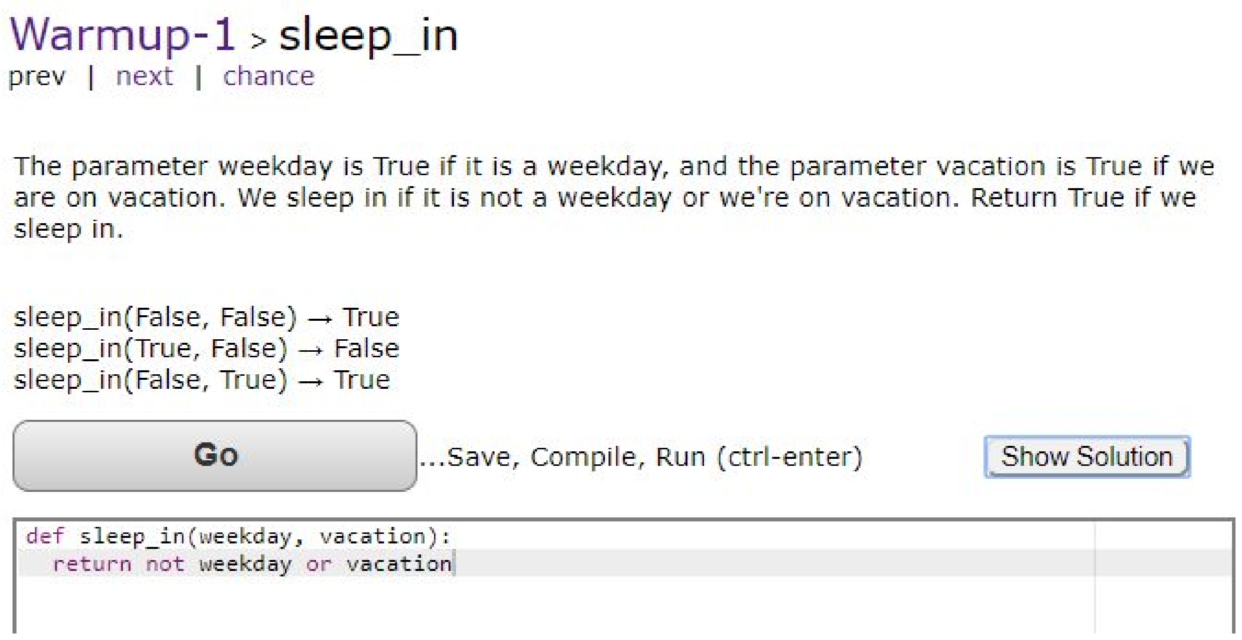


Warmup-1 > sleep_in

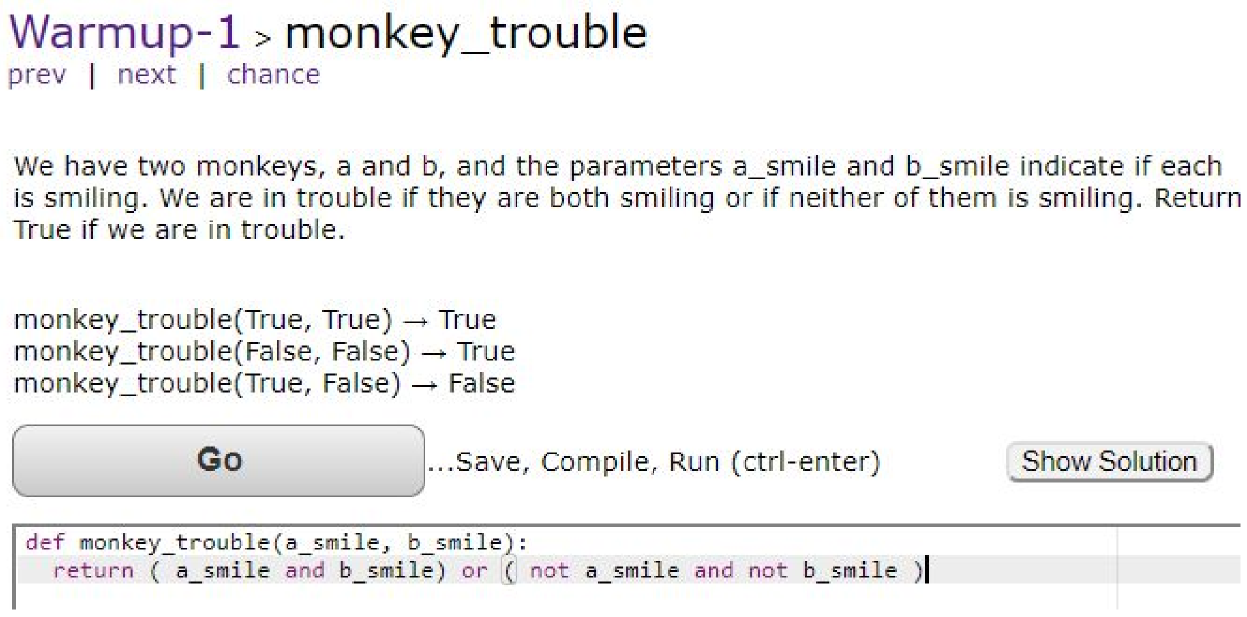
prev | next | chance

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

Go ...Save, Compile, Run (ctrl-enter) Show Solution

```
def sleep_in(weekday, vacation):
  return not weekday or vacation
```



Warmup-1 > monkey_trouble

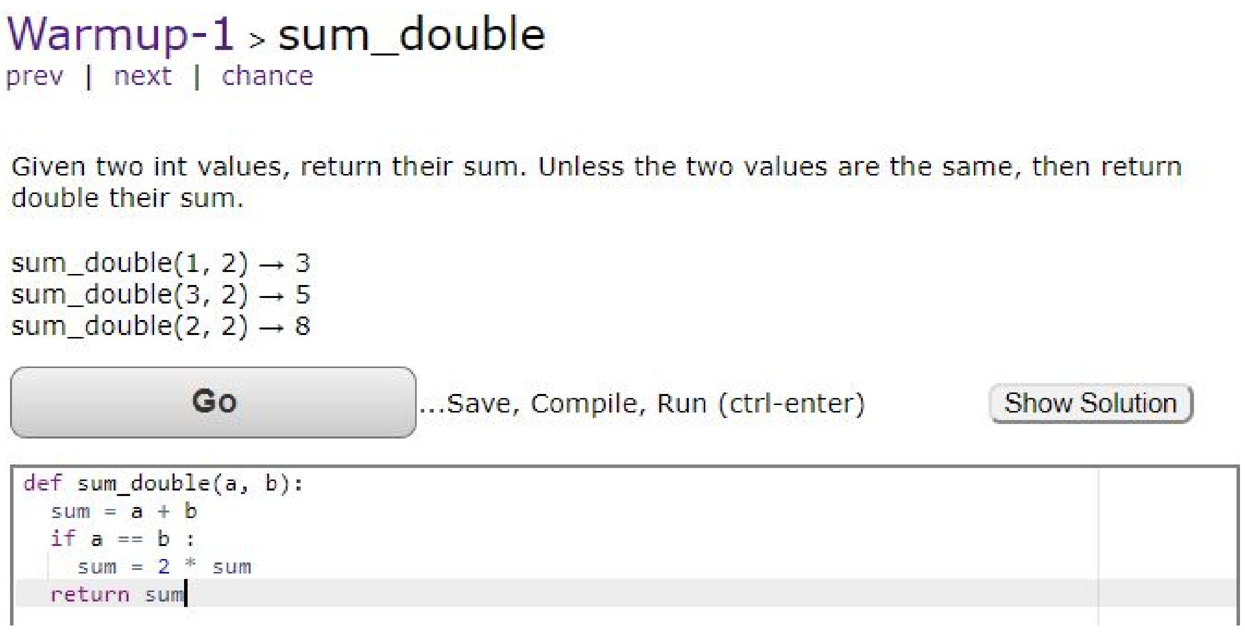
prev | next | chance

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

Go ...Save, Compile, Run (ctrl-enter) Show Solution

```
def monkey_trouble(a_smile, b_smile):
  return ( a_smile and b_smile) or ( not a_smile and not b_smile )
```



Warmup-1 > sum_double

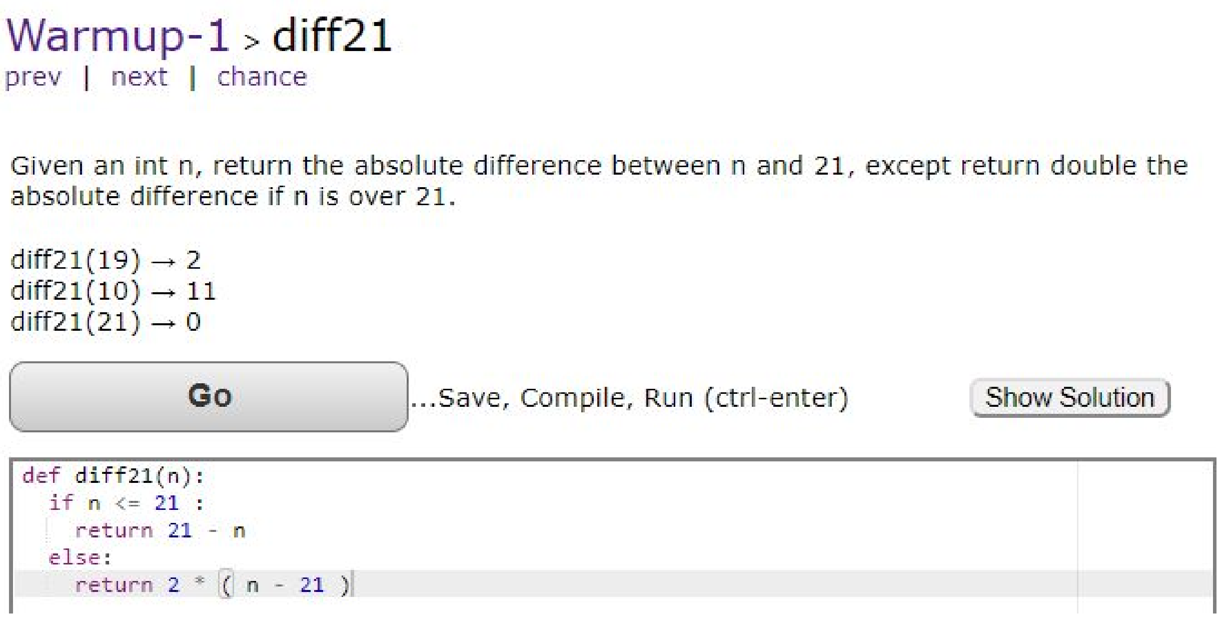
prev | next | chance

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

...Save, Compile, Run (ctrl-enter)

```python
def sum_double(a, b):
  sum = a + b
  if a == b :
    sum = 2 * sum
  return sum
```



Warmup-1 > diff21

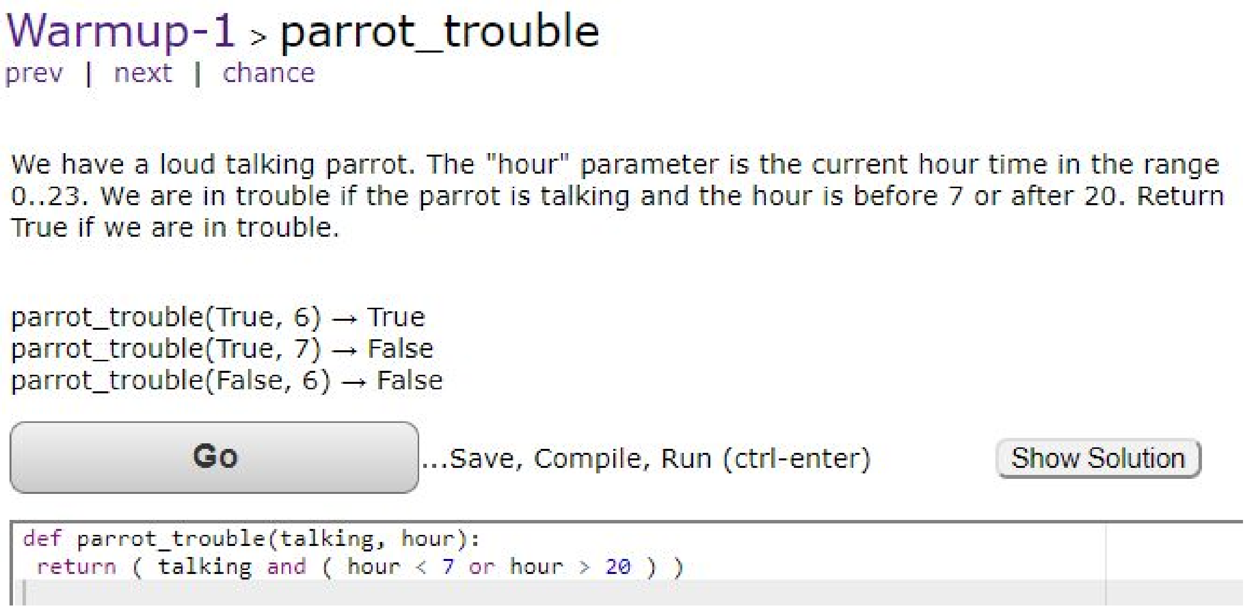
prev | next | chance

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

...Save, Compile, Run (ctrl-enter)

```python
def diff21(n):
  if n <= 21 :
    return 21 - n
  else:
    return 2 * ( n - 21 )
```



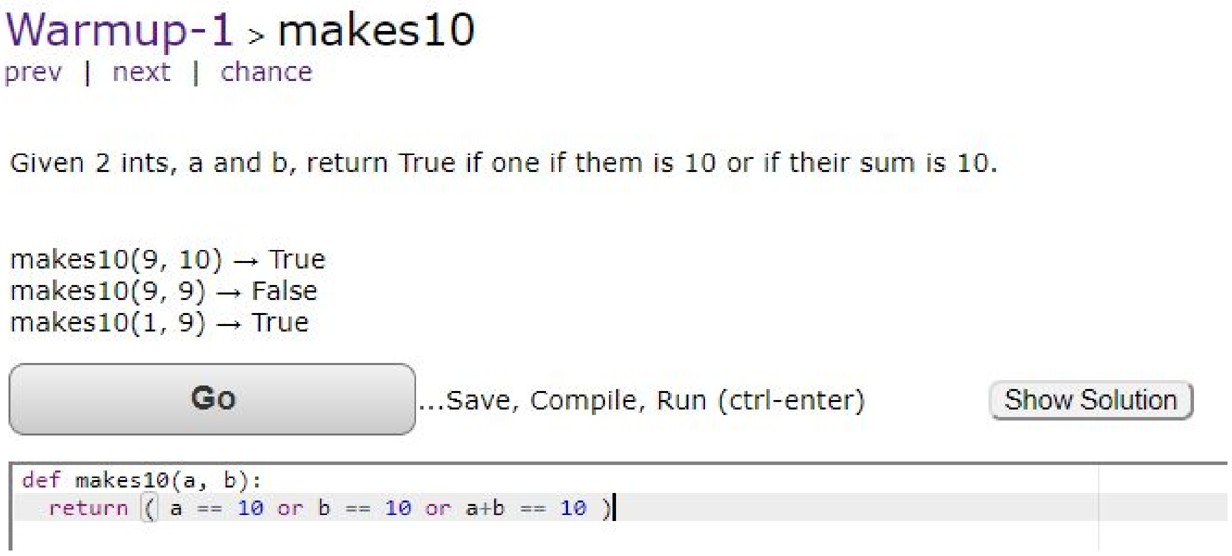
Warmup-1 > parrot_trouble

prev | next | chance

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 and ( hour < 7 or hour > 20 ) )
```



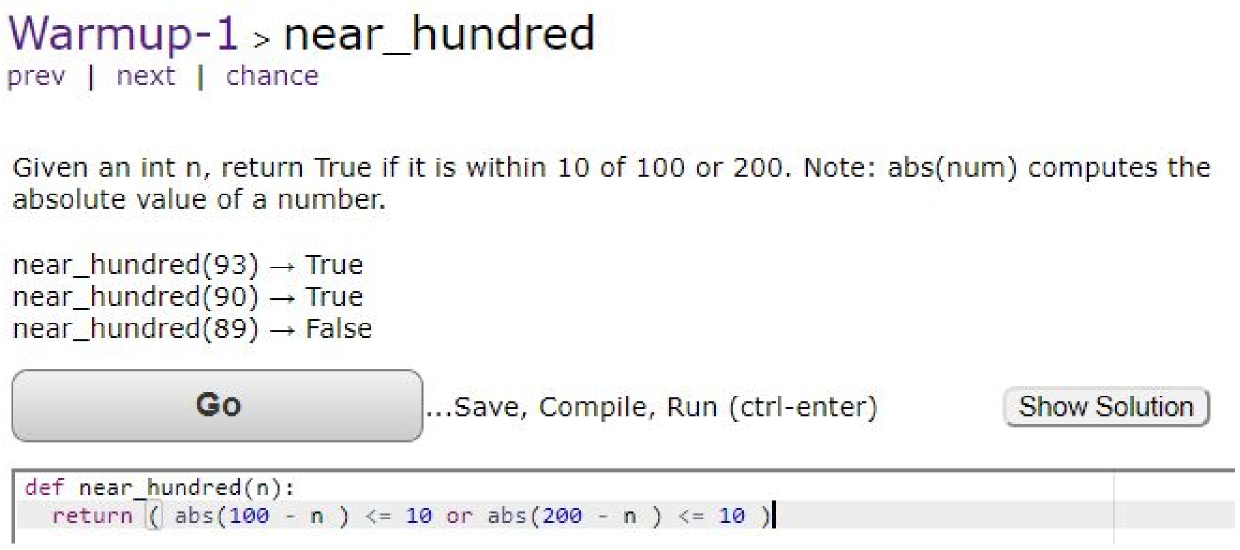
Warmup-1 > makes10

prev | next | chance

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 )
```



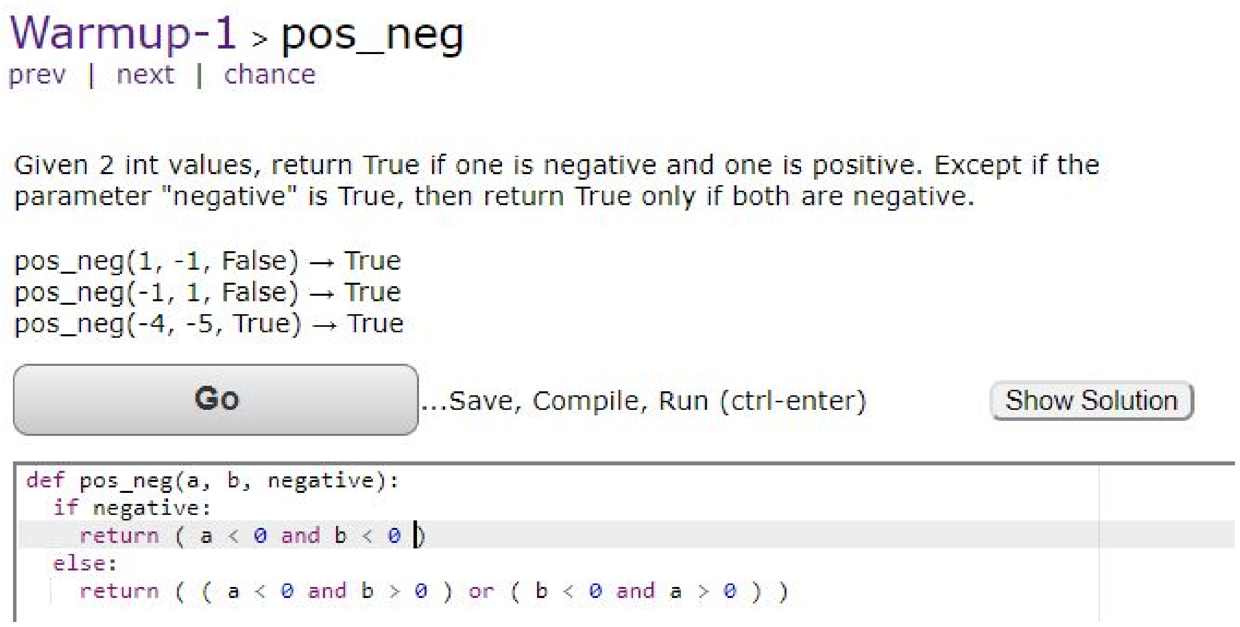
Warmup-1 > near_hundred

prev | next | chance

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 )
```



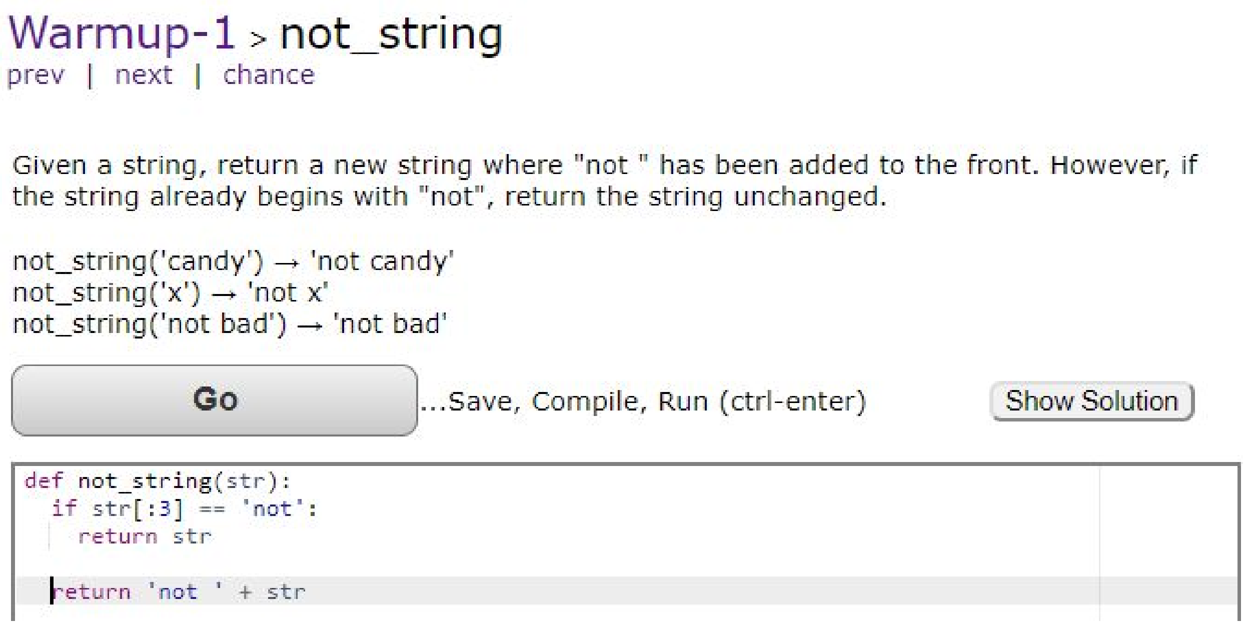
Warmup-1 > pos_neg

prev | next | chance

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:
    return ( a < 0 and b < 0 )
  else:
    return ( ( a < 0 and b > 0 ) or ( b < 0 and a > 0 ) )
```



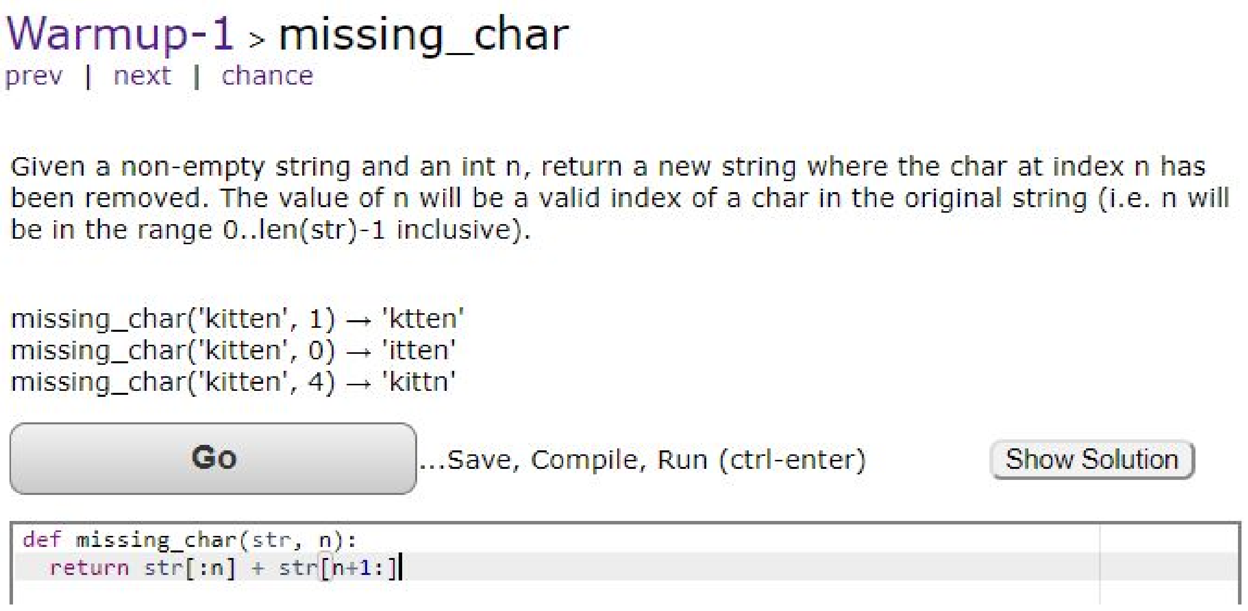
Warmup-1 > not_string

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 str[:3] == 'not':
    return str

  return 'not ' + str
```

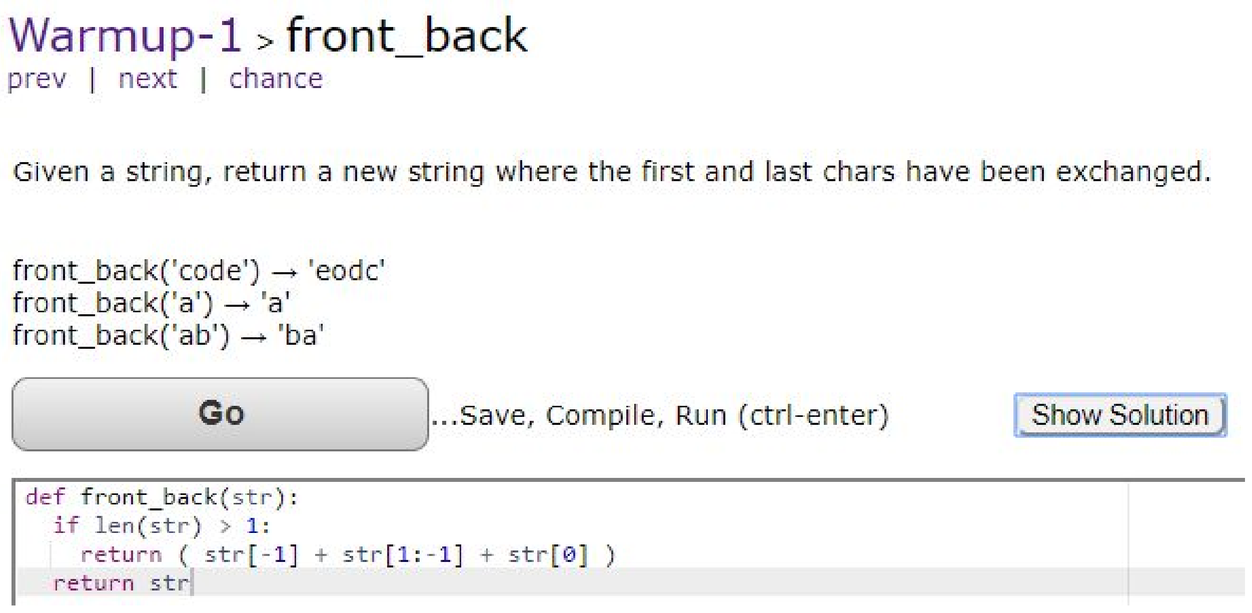


Warmup-1 > missing_char

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[:n] + str[n+1:]
```



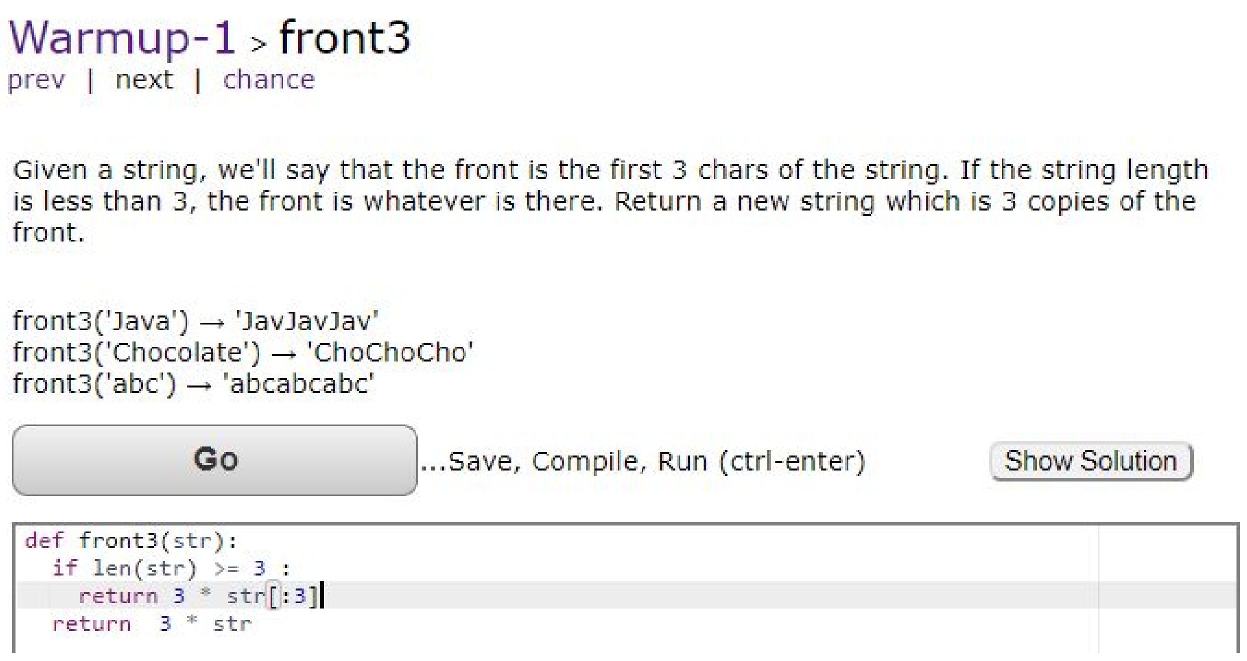
Warmup-1 > front_back

prev | next | chance

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[-1] + str[1:-1] + str[0] )
  return str
```



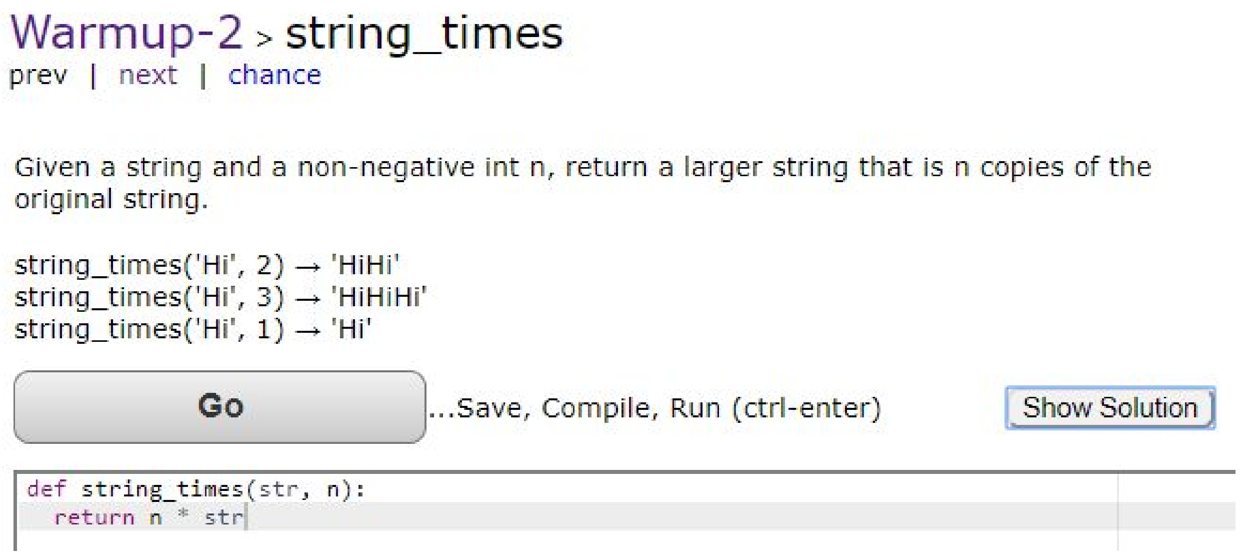
Warmup-1 > front3

prev | next | chance

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 3 * str[:3]
  return  3 * str
```



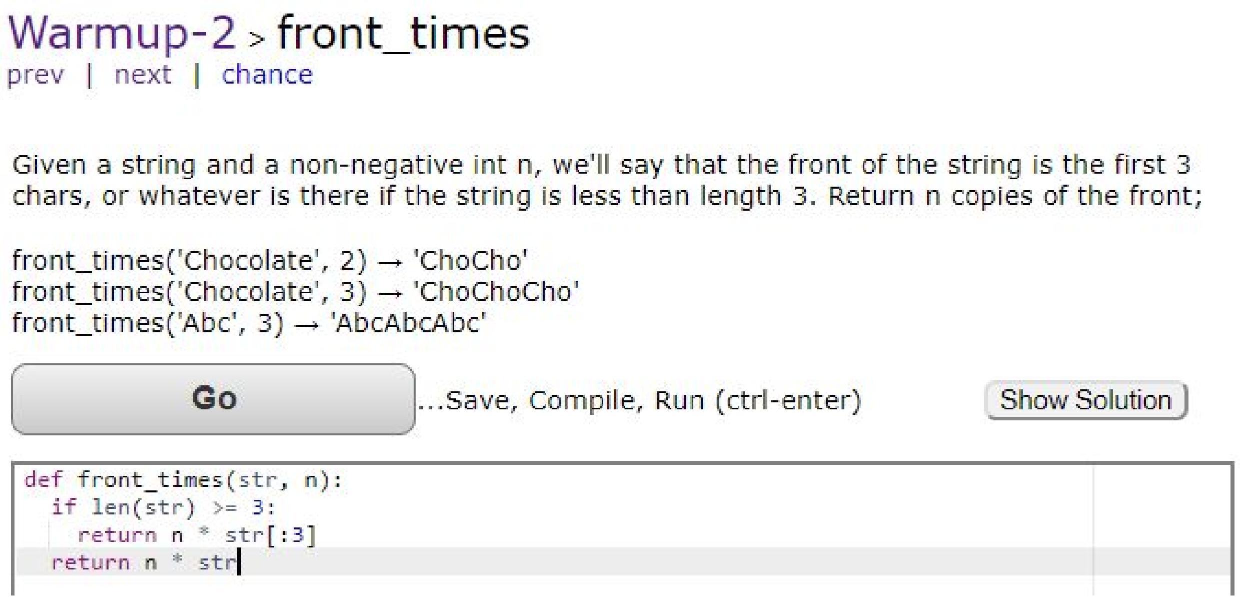
Warmup-2 > string_times

prev | next | chance

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):
  return n * str
```



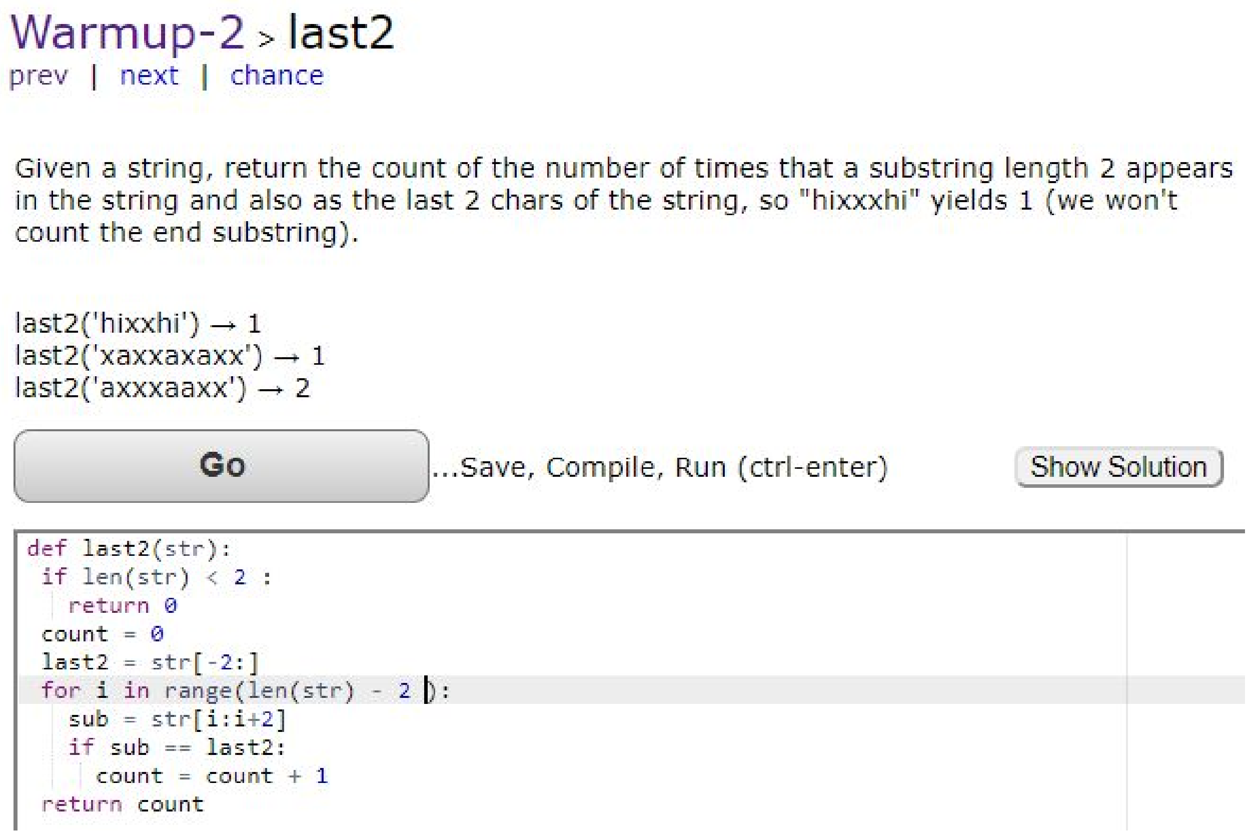
Warmup-2 > front_times

prev | next | chance

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):
  if len(str) >= 3:
    return n * str[:3]
  return n * str
```



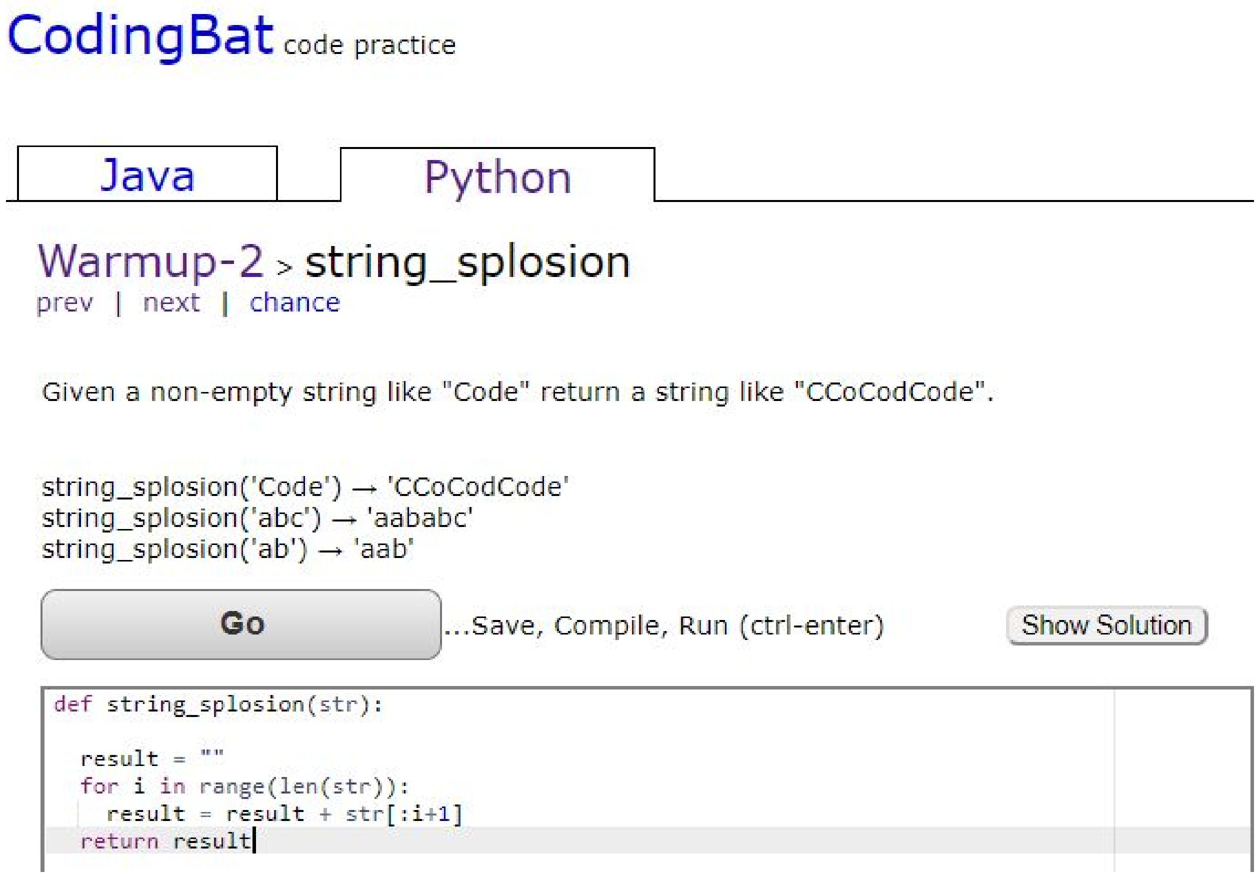
Warmup-2 > last2

prev | next | chance

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
 count = 0
 last2 = str[-2:]
 for i in range(len(str) - 2 ):
   sub = str[i:i+2]
   if sub == last2:
     count = count + 1
 return count
```



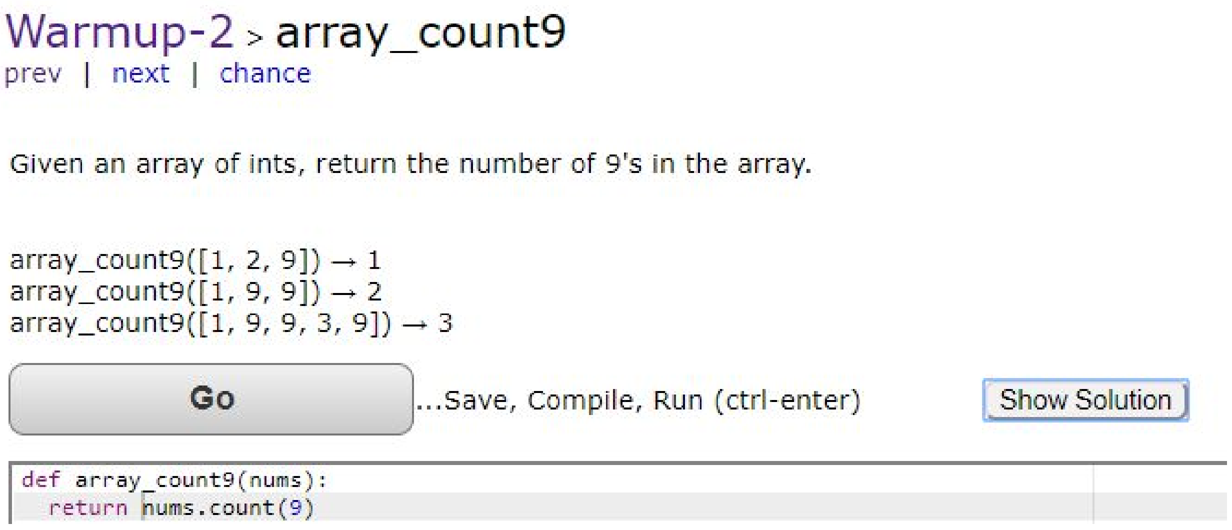
Warmup-2 > string_splosion

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):

  result = ""
  for i in range(len(str)):
    result = result + str[:i+1]
  return result
```

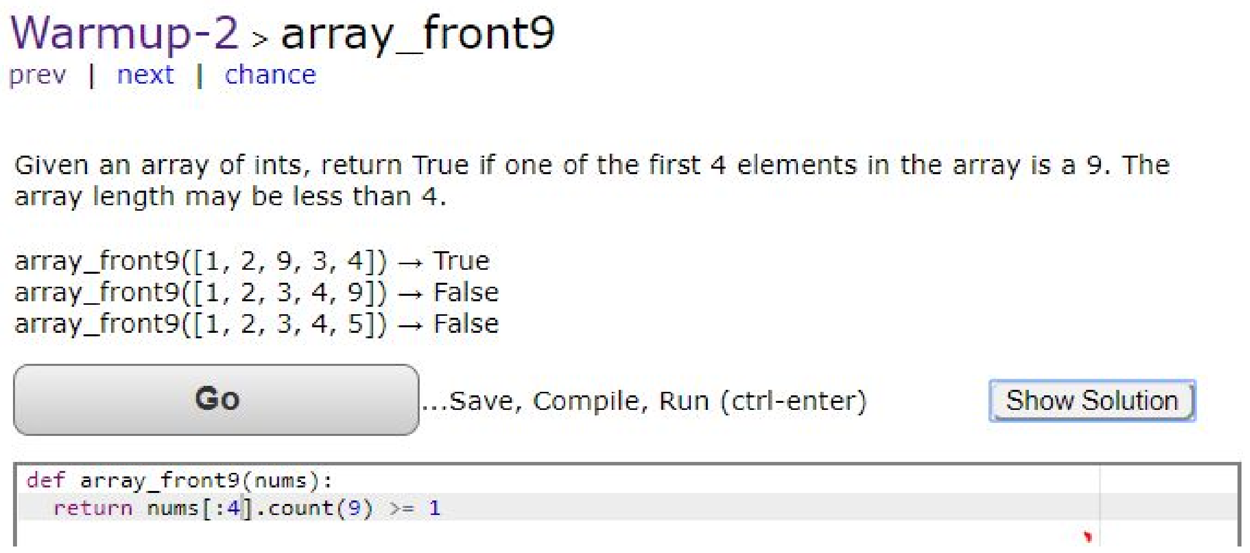


Warmup-2 > array_count9

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):
  return nums.count(9)
```



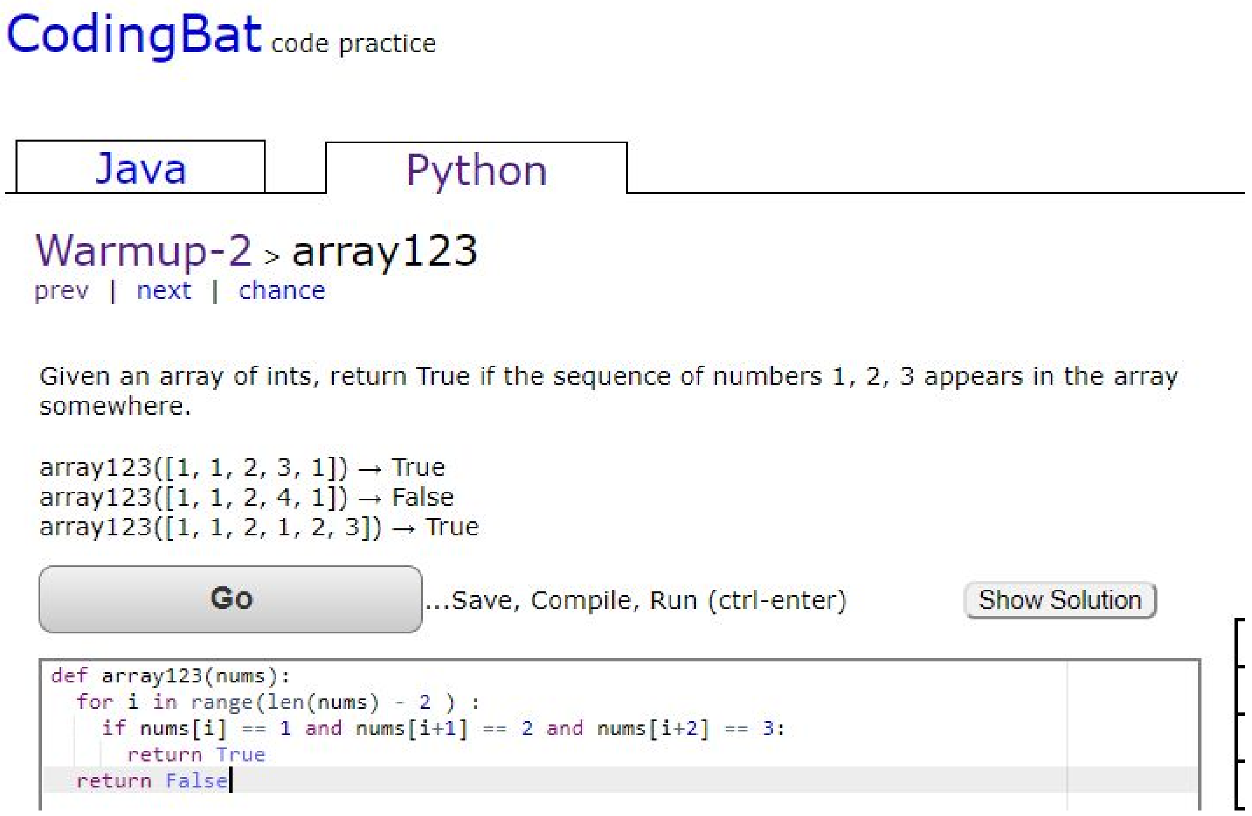
Warmup-2 > array_front9

prev | next | chance

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

```python
def array_front9(nums):
  return nums[:4].count(9) >= 1
```



CodingBat code practice

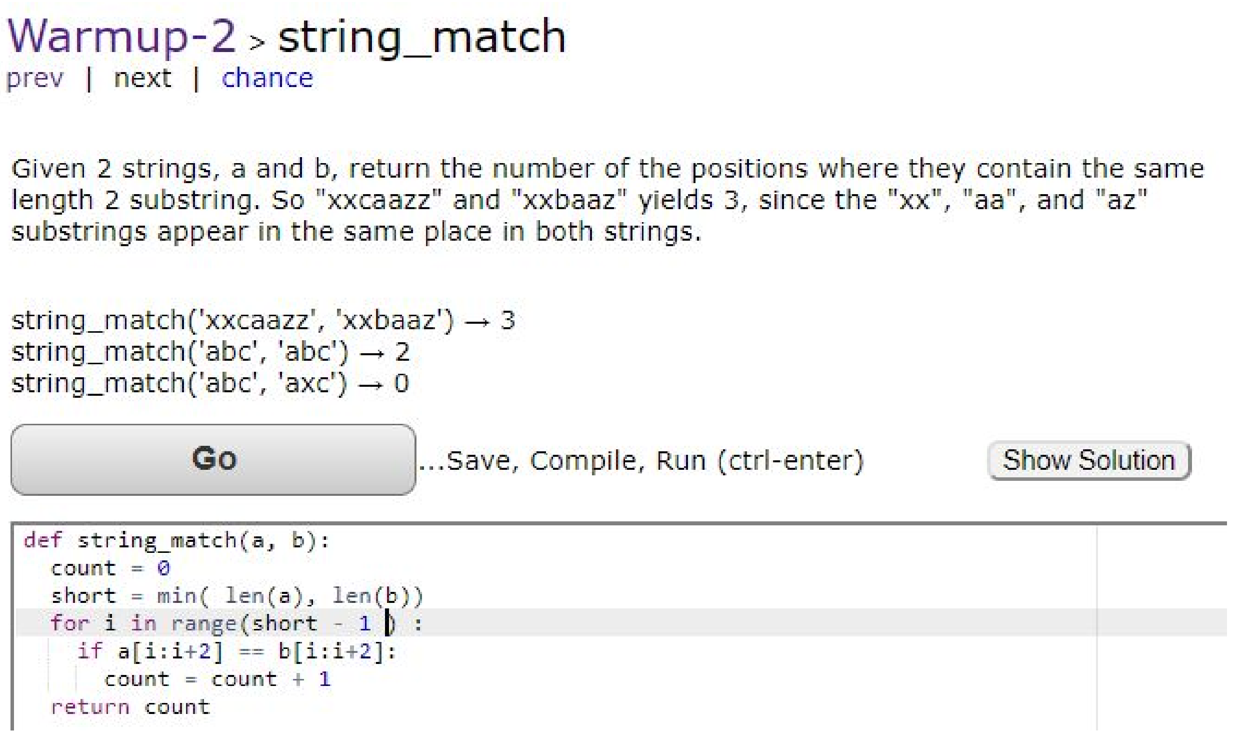
Warmup-2 > array123

prev | next | chance

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

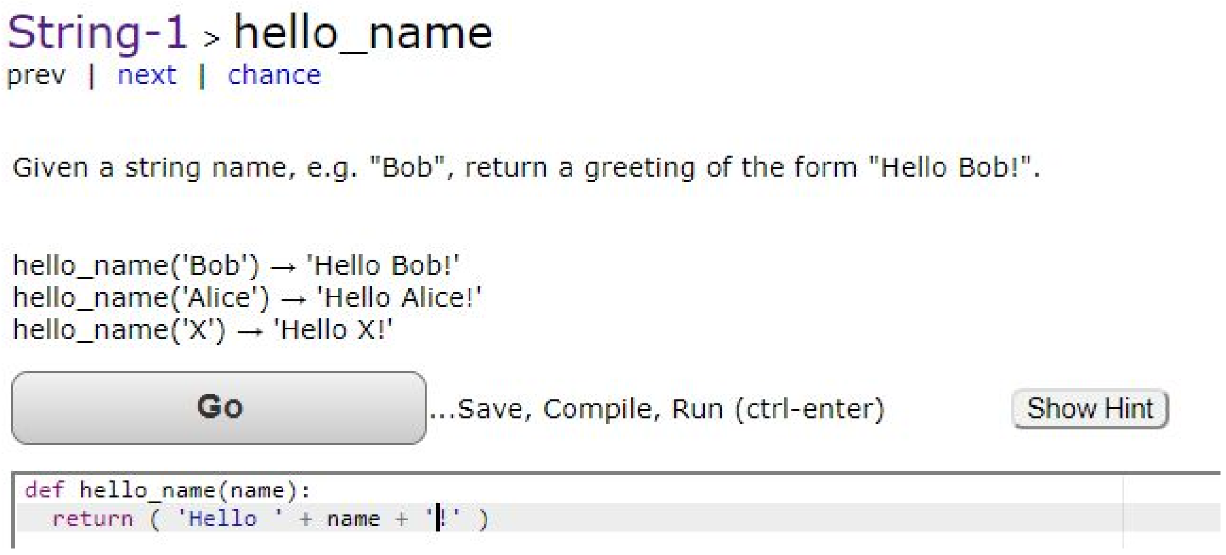
```python
def array123(nums):
  for i in range(len(nums) - 2 ) :
    if nums[i] == 1 and nums[i+1] == 2 and nums[i+2] == 3:
      return True
  return False
```


Warmup-2 > string_match
prev | next | chance

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

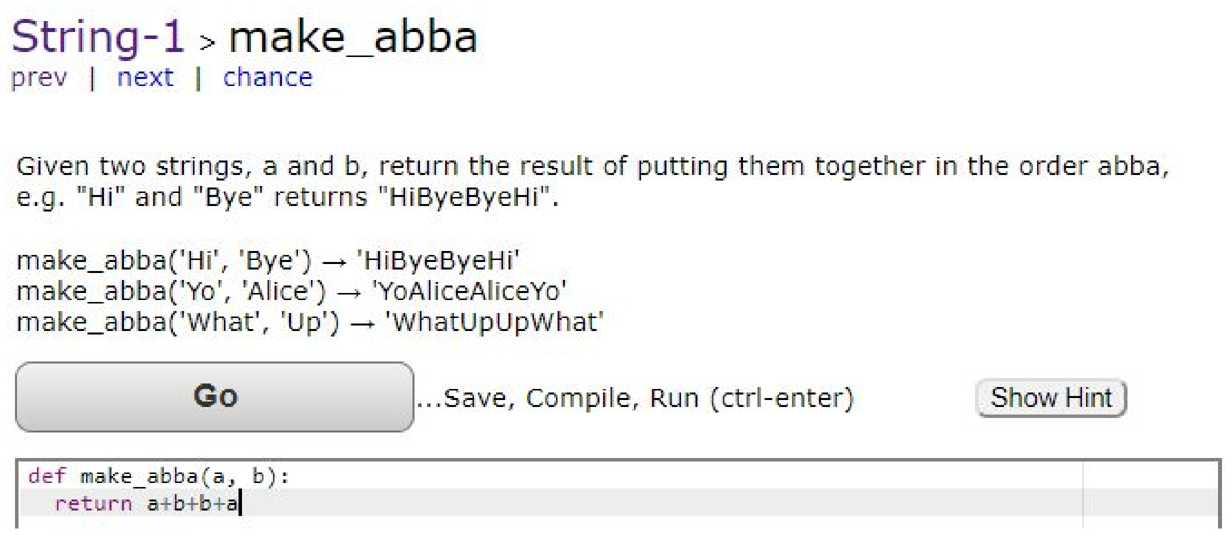
```
def string_match(a, b):
  count = 0
  short = min( len(a), len(b))
  for i in range(short - 1 ) :
    if a[i:i+2] == b[i:i+2]:
      count = count + 1
  return count
```


String-1 > hello_name
prev | next | chance

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 + '!' )
```

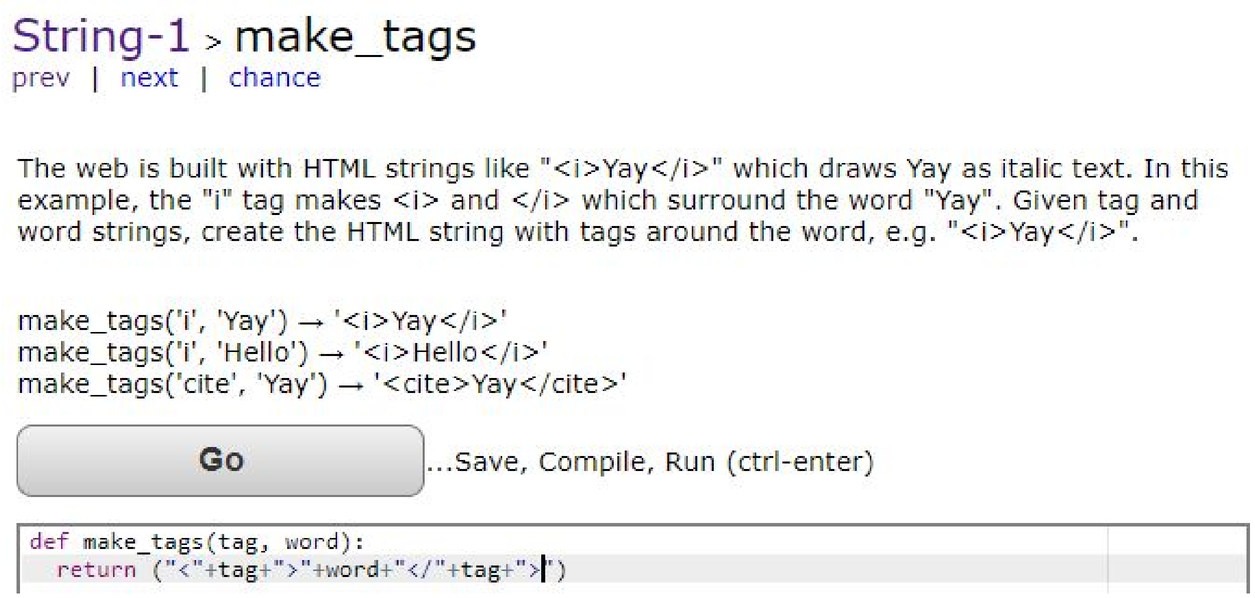


String-1 > make_abba

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
```

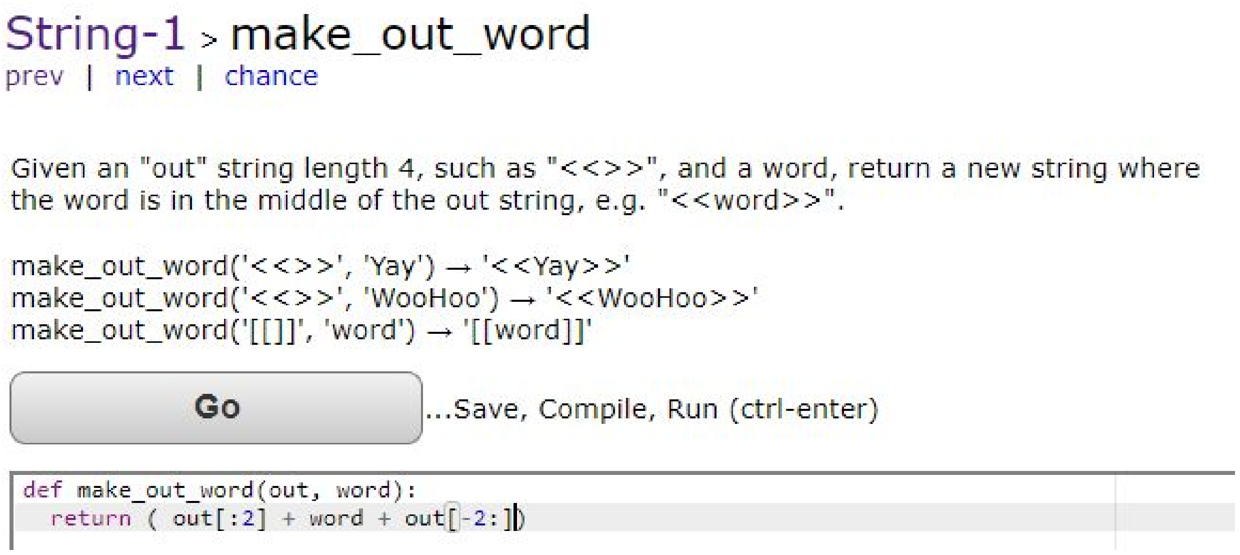


String-1 > make_tags

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+">")
```



String-1 > make_out_word

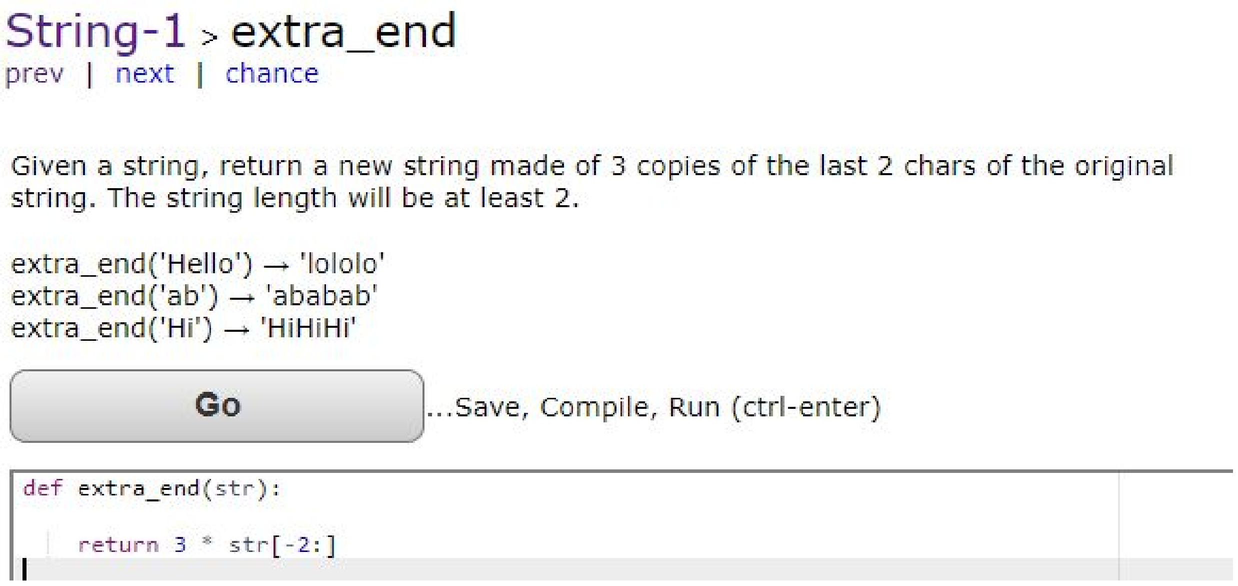
prev | next | chance

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]]'

...Save, Compile, Run (ctrl-enter)

```
def make_out_word(out, word):
  return ( out[:2] + word + out[-2:])
```



String-1 > extra_end

prev | next | chance

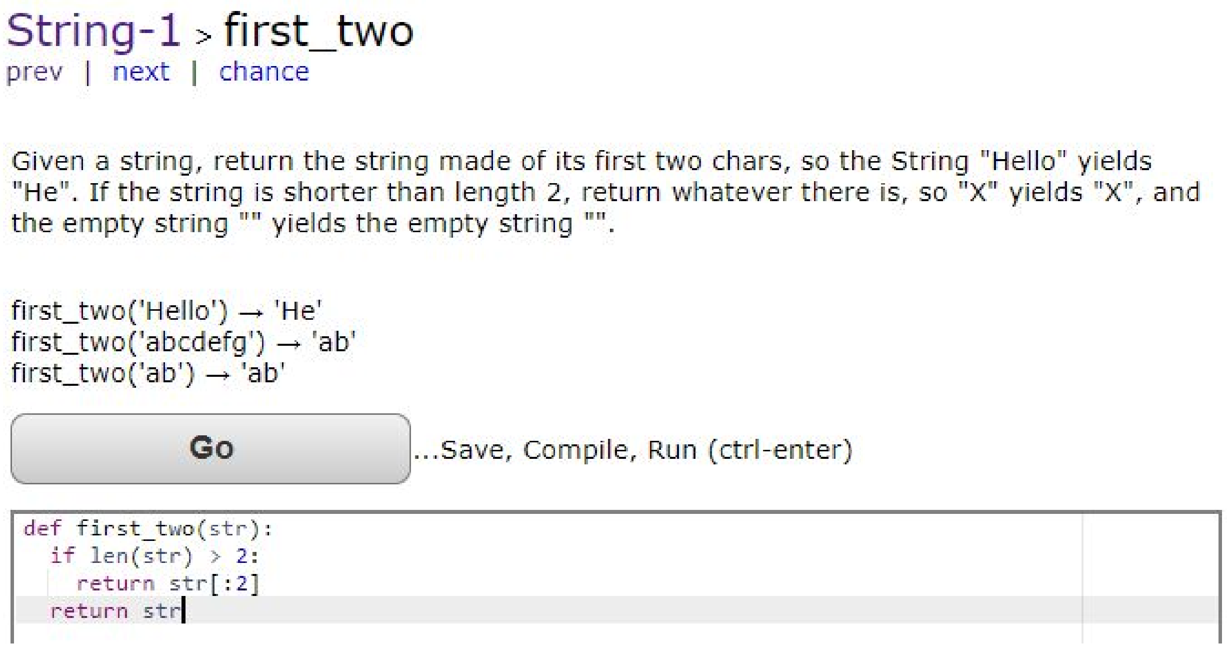
Given a string, return a new string made of 3 copies of the last 2 chars of the original string. The string length will be at least 2.

extra_end('Hello') → 'lololo'
extra_end('ab') → 'ababab'
extra_end('Hi') → 'HiHiHi'

...Save, Compile, Run (ctrl-enter)

```
def extra_end(str):

    return 3 * str[-2:]
```



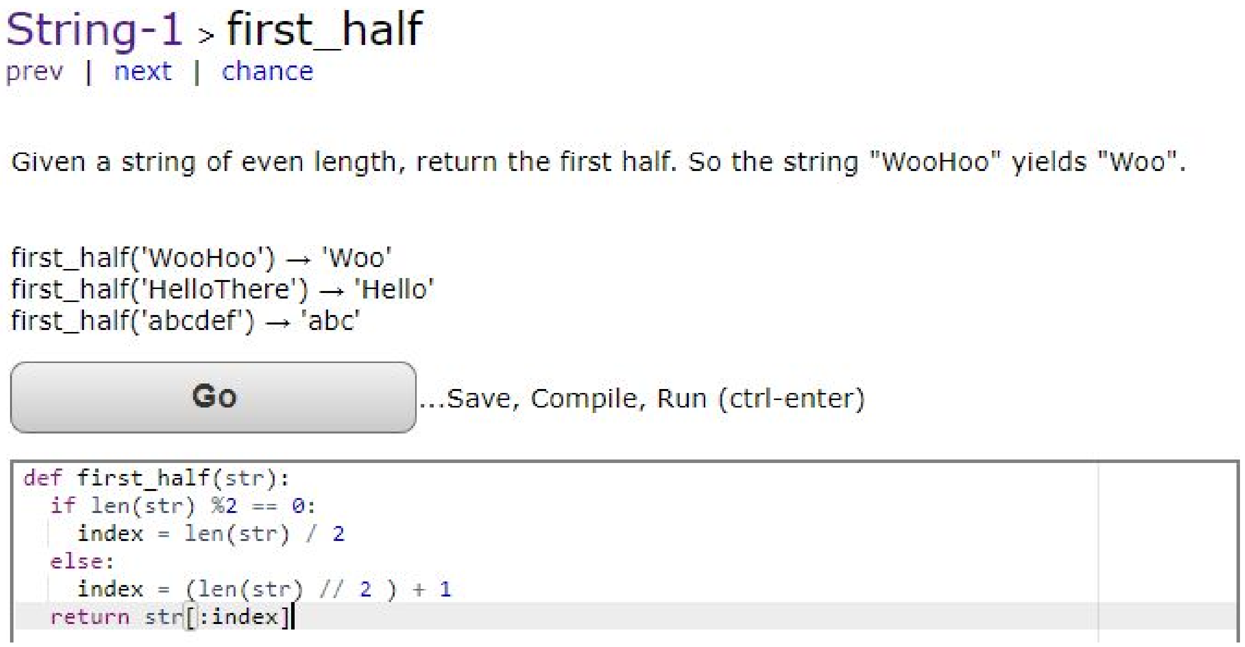
String-1 > first_two

prev | next | chance

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):
  if len(str) > 2:
    return str[:2]
  return str
```



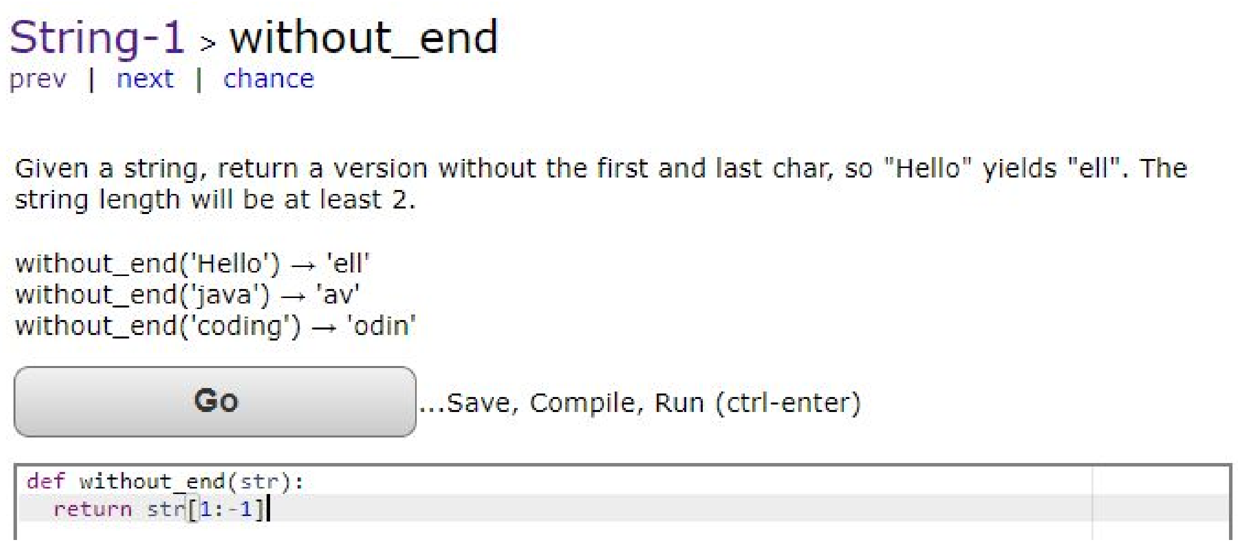
String-1 > first_half

prev | next | chance

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):
  if len(str) %2 == 0:
    index = len(str) / 2
  else:
    index = (len(str) // 2 ) + 1
  return str[:index]
```



String-1 > without_end

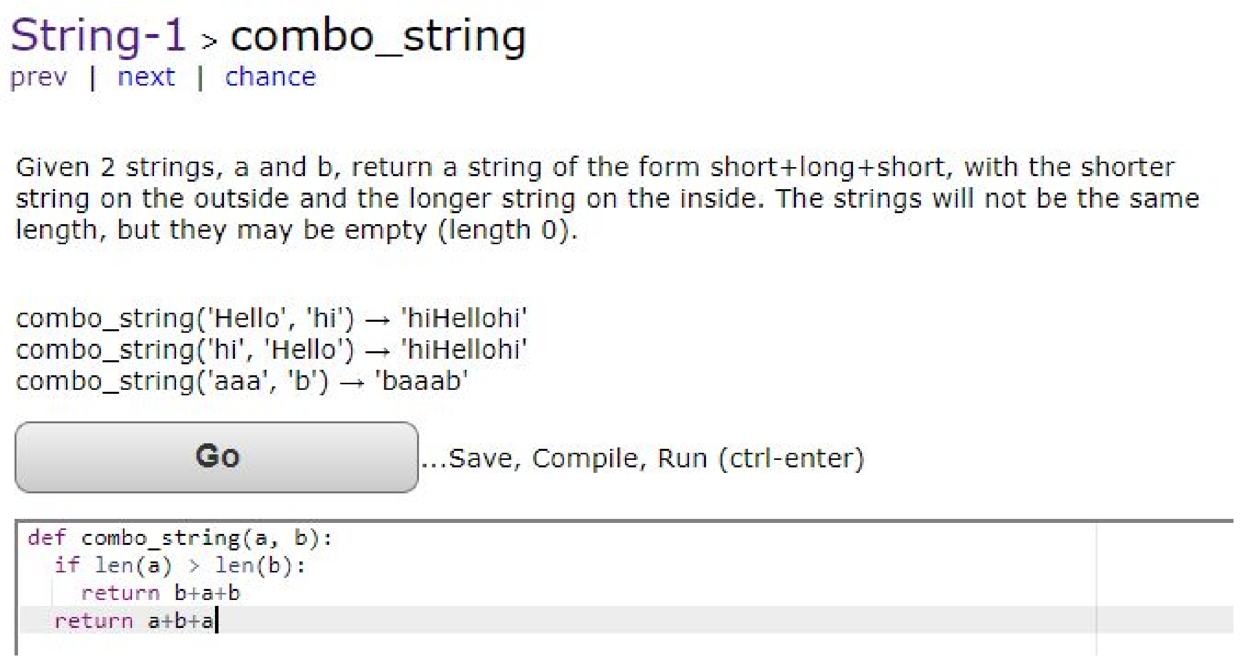
prev | next | chance

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'

...Save, Compile, Run (ctrl-enter)

```
def without_end(str):
  return str[1:-1]
```



String-1 > combo_string

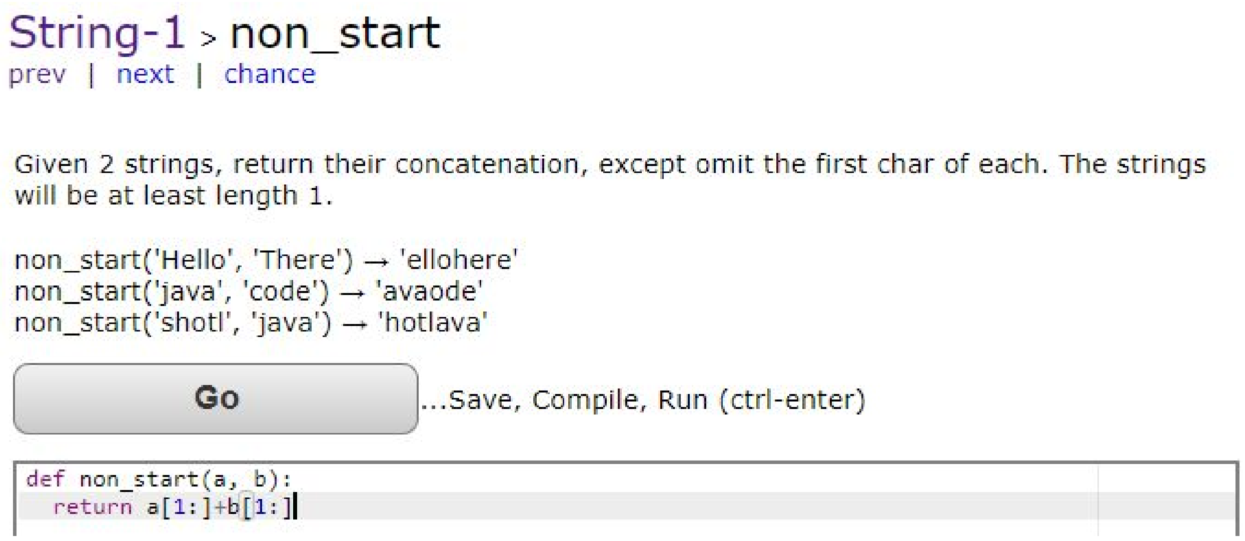
prev | next | chance

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'

...Save, Compile, Run (ctrl-enter)

```
def combo_string(a, b):
  if len(a) > len(b):
    return b+a+b
  return a+b+a
```



String-1 > non_start

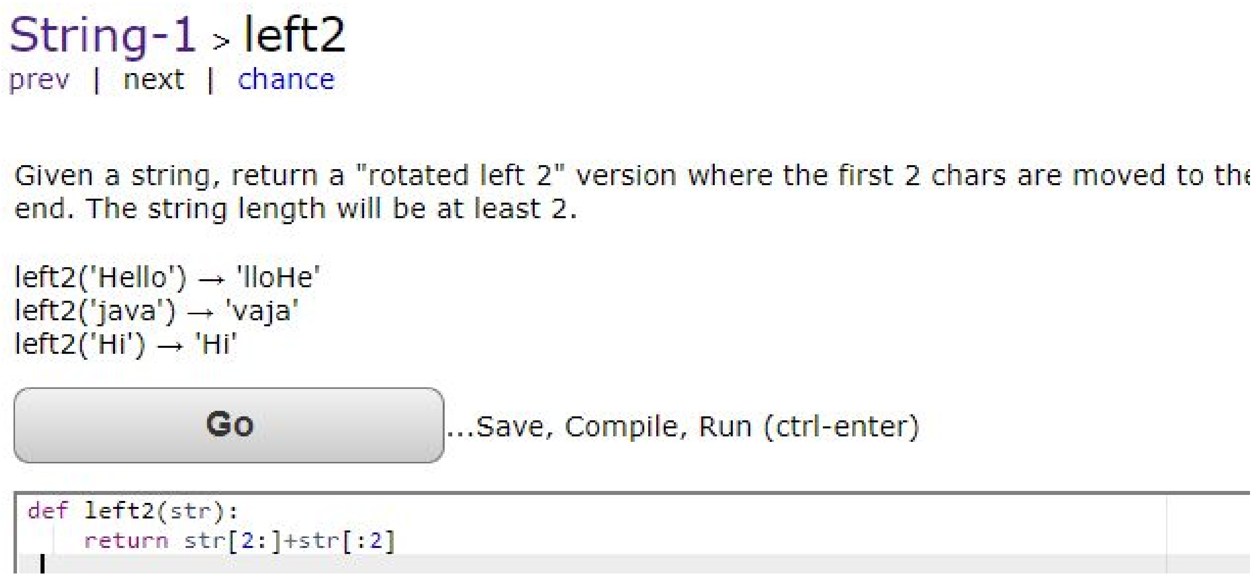
prev | next | chance

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'

...Save, Compile, Run (ctrl-enter)

```
def non_start(a, b):
  return a[1:]+b[1:]
```



String-1 > left2

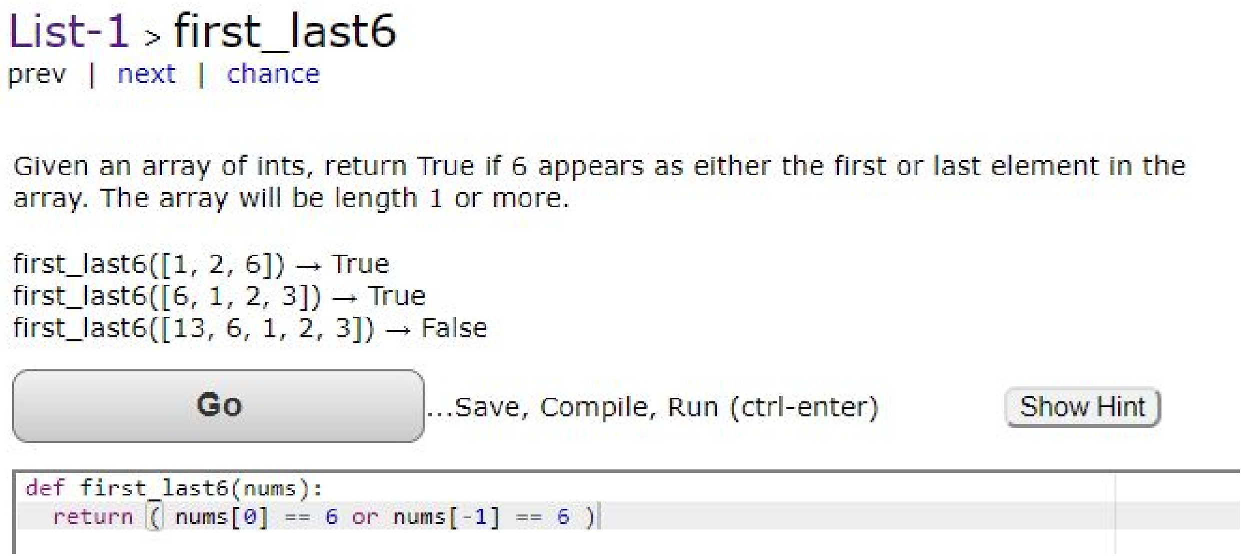
prev | next | chance

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'

...Save, Compile, Run (ctrl-enter)

```
def left2(str):
    return str[2:]+str[:2]
```



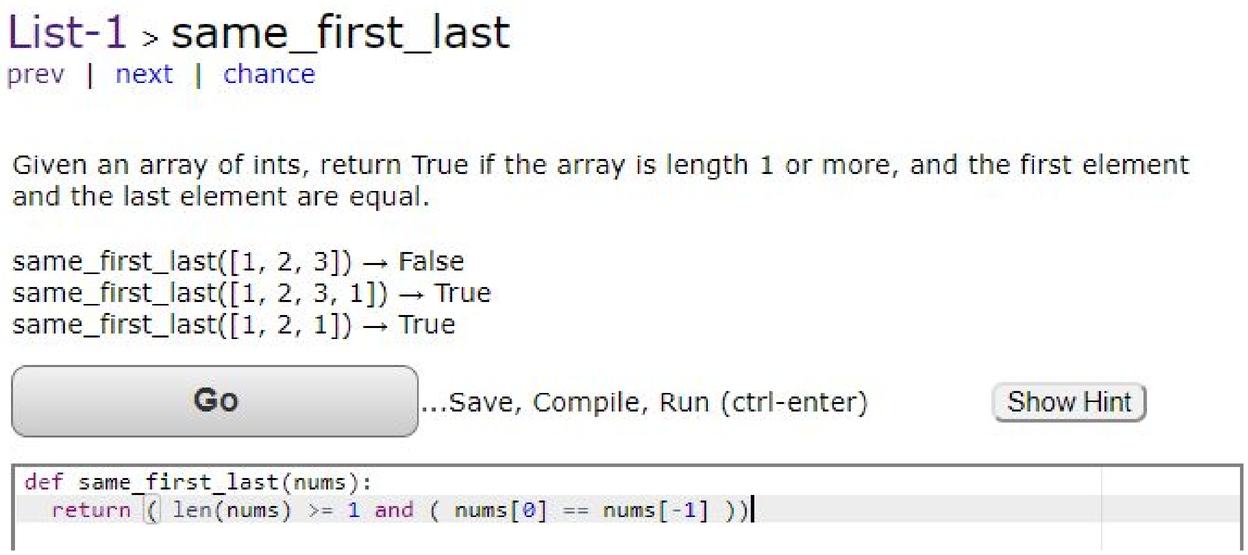
List-1 > first_last6

prev | next | chance

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):
  return ( nums[0] == 6 or nums[-1] == 6 )
```



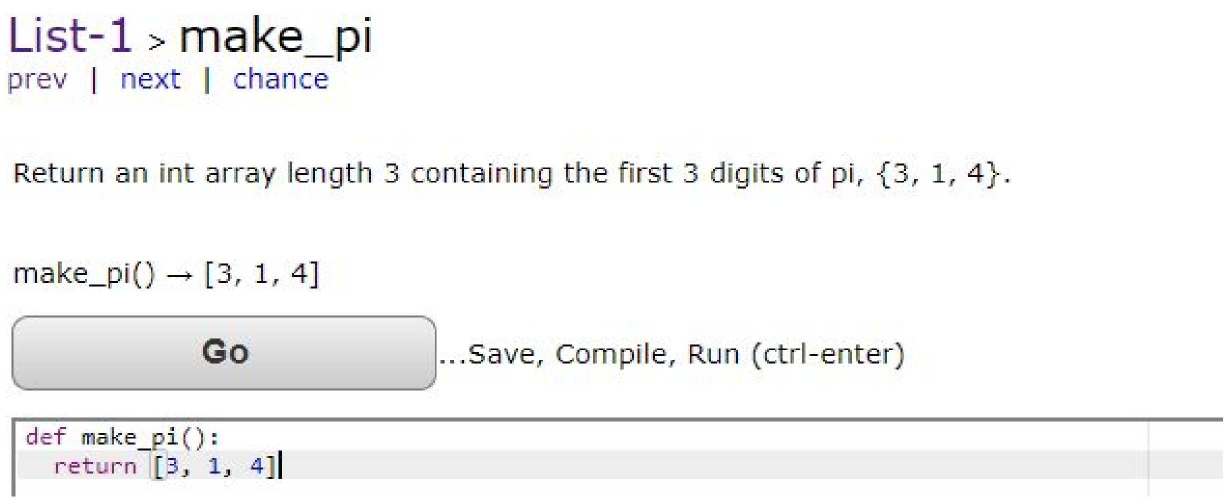
List-1 > same_first_last

prev | next | chance

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[-1] ))
```



List-1 > make_pi

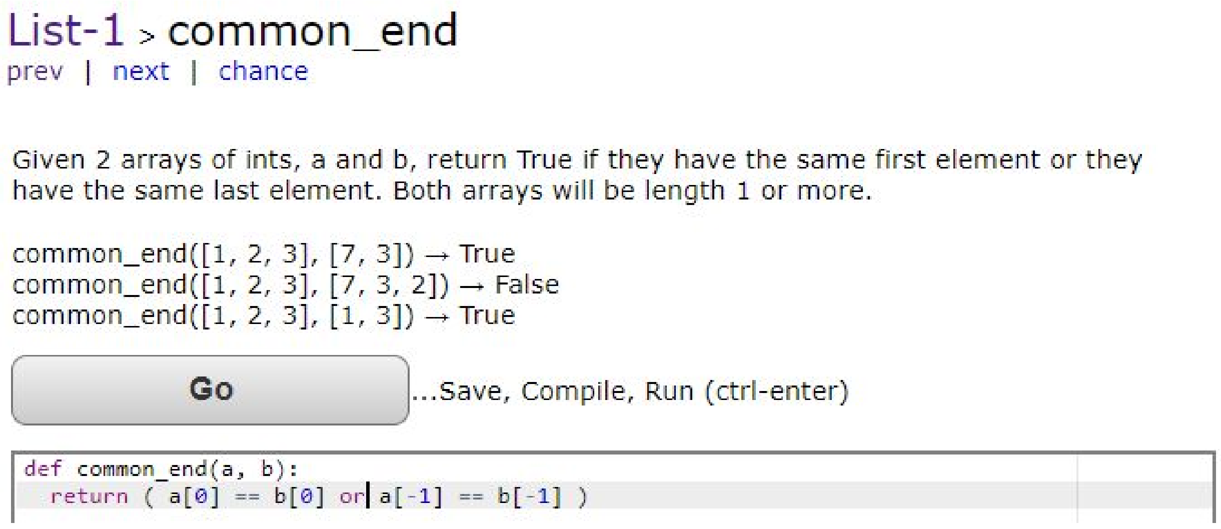
prev | next | chance

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

make_pi() → [3, 1, 4]

...Save, Compile, Run (ctrl-enter)

```
def make_pi():
  return [3, 1, 4]
```



List-1 > common_end

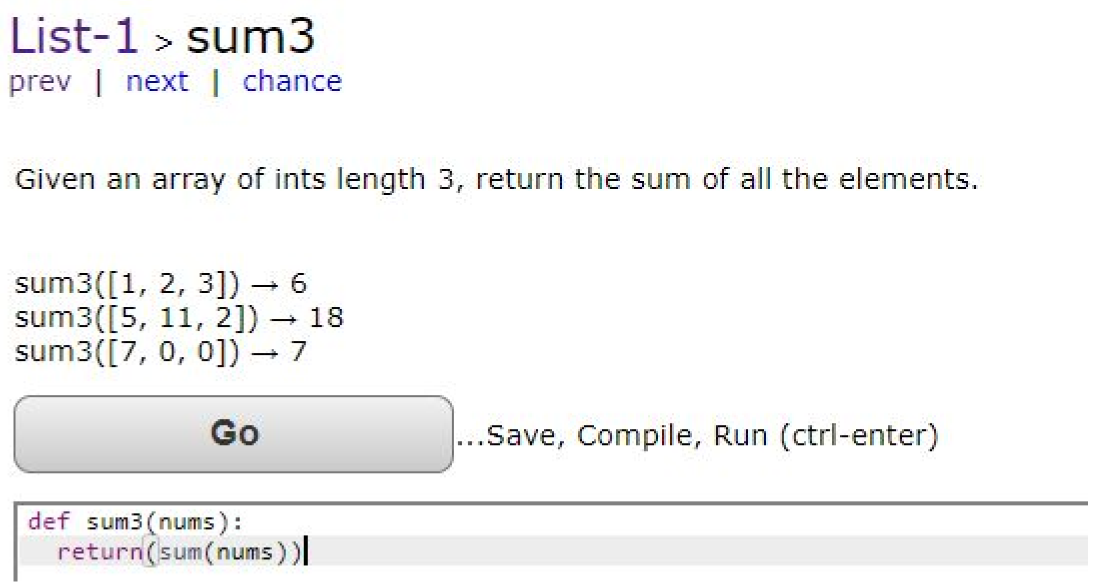
prev | next | chance

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

...Save, Compile, Run (ctrl-enter)

```
def common_end(a, b):
  return ( a[0] == b[0] or a[-1] == b[-1] )
```

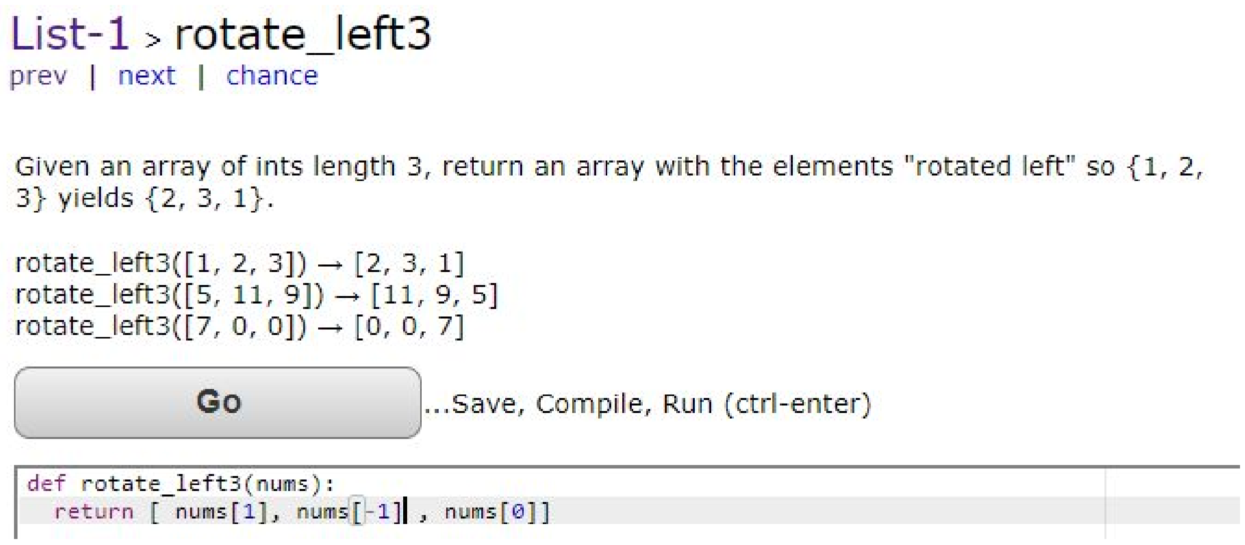


List-1 > sum3

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):
  return(sum(nums))
```

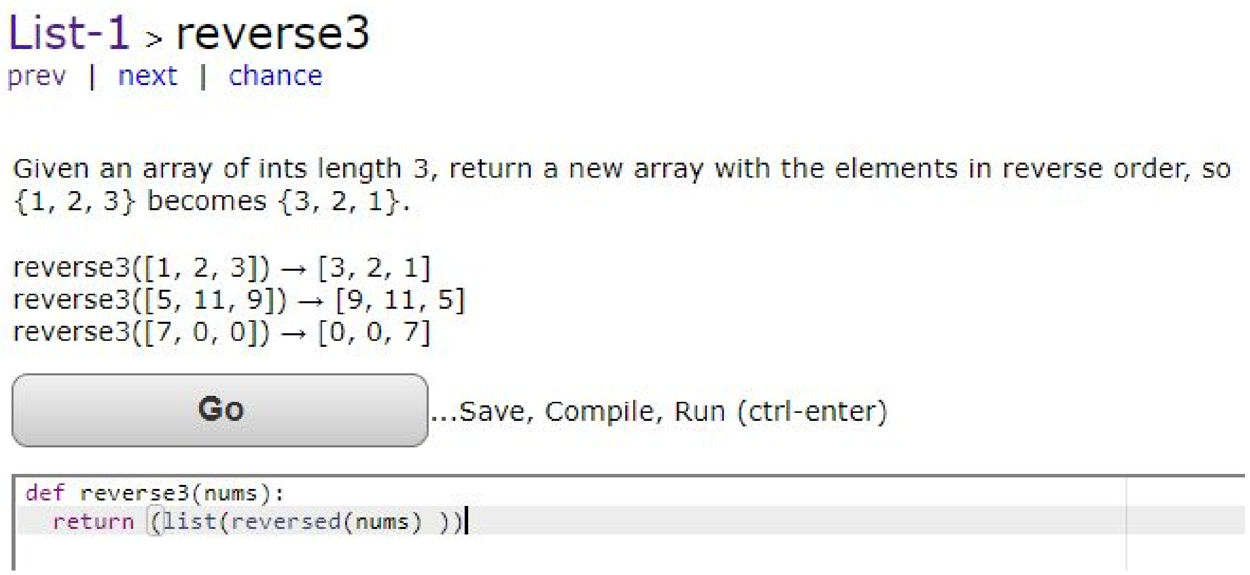


List-1 > rotate_left3

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[-1] , nums[0]]
```



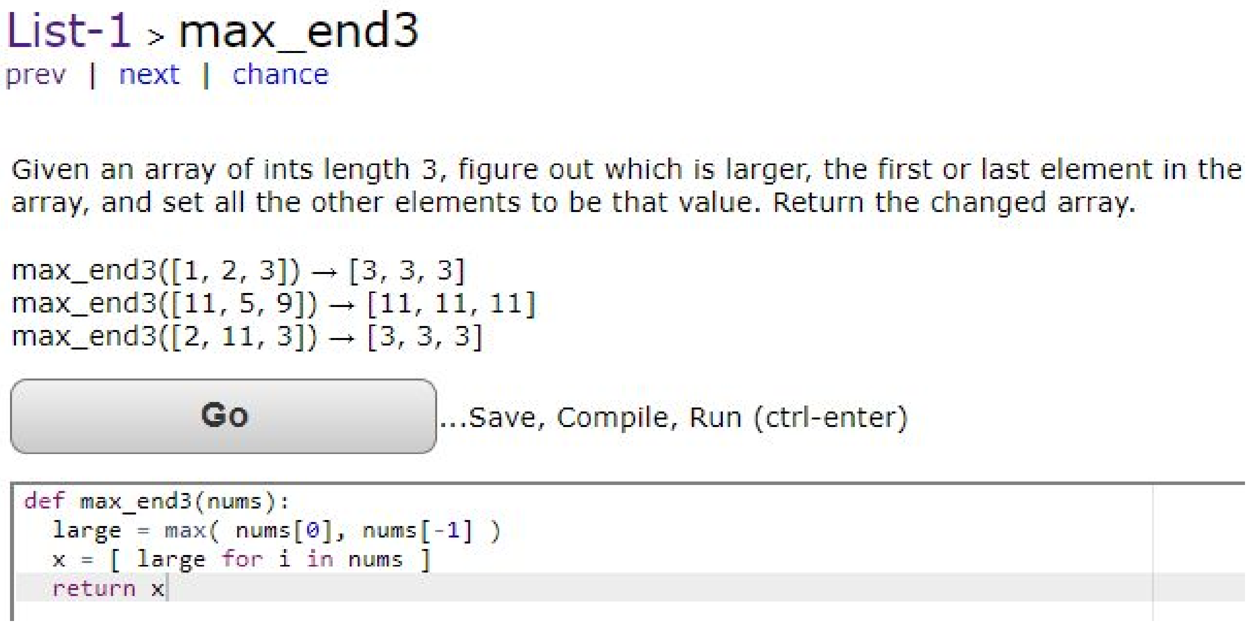
List-1 > reverse3

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]

...Save, Compile, Run (ctrl-enter)

```
def reverse3(nums):
  return (list(reversed(nums) ))
```



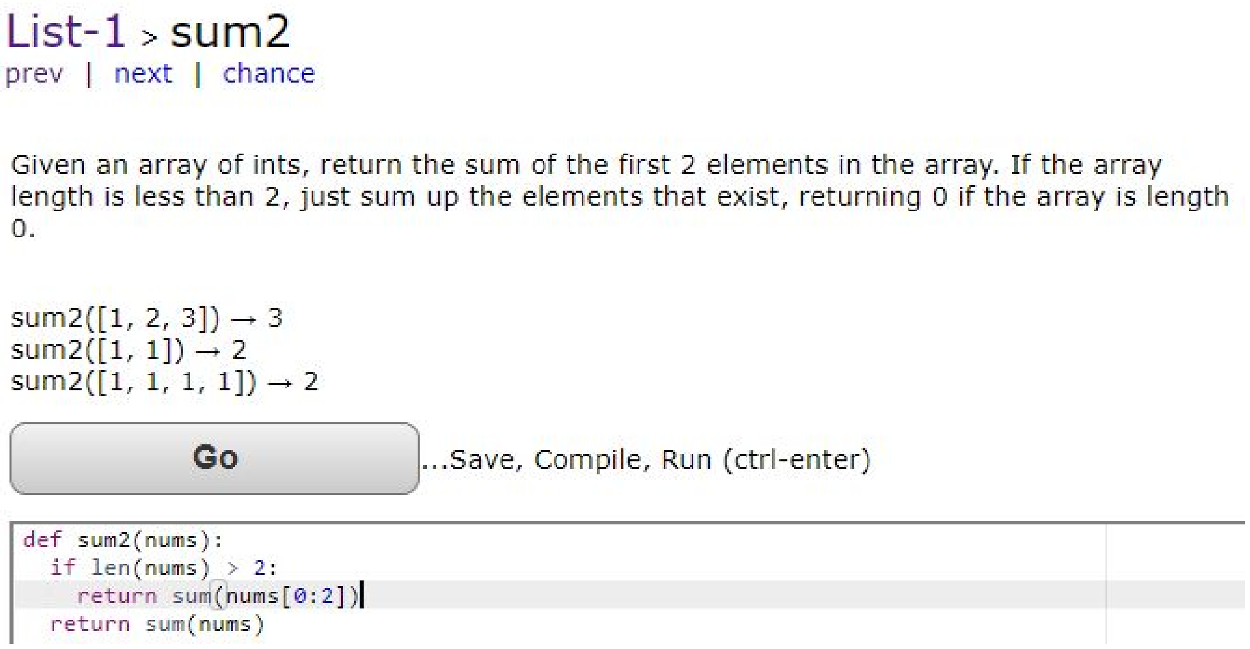
List-1 > max_end3

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]

...Save, Compile, Run (ctrl-enter)

```
def max_end3(nums):
  large = max( nums[0], nums[-1] )
  x = [ large for i in nums ]
  return x
```



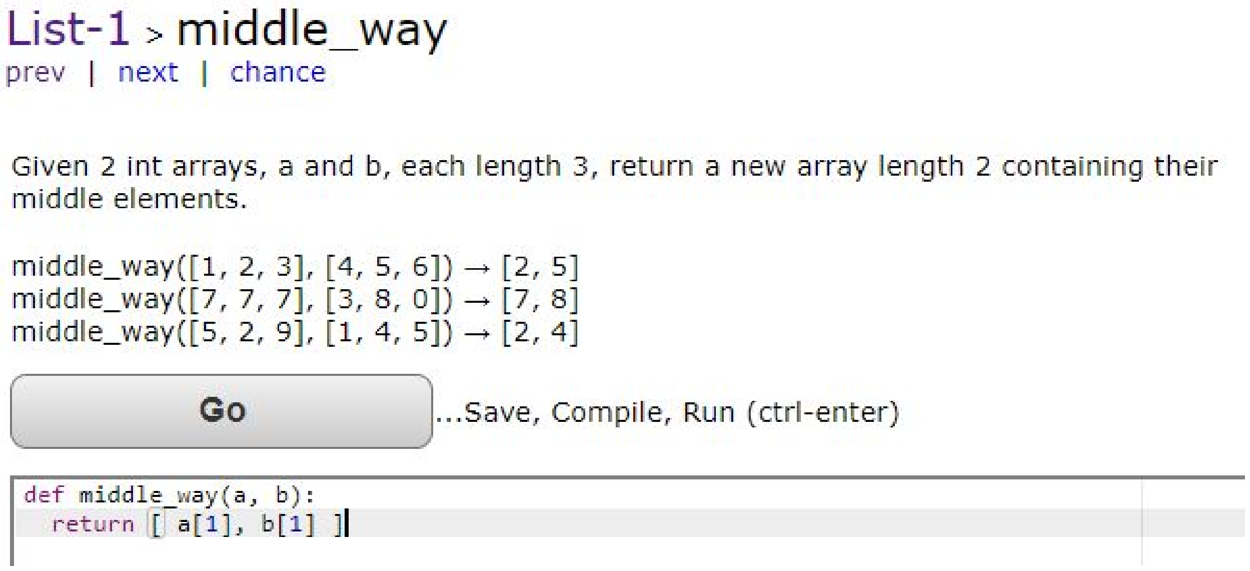
List-1 > sum2

prev | next | chance

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

```python
def sum2(nums):
  if len(nums) > 2:
    return sum(nums[0:2])
  return sum(nums)
```



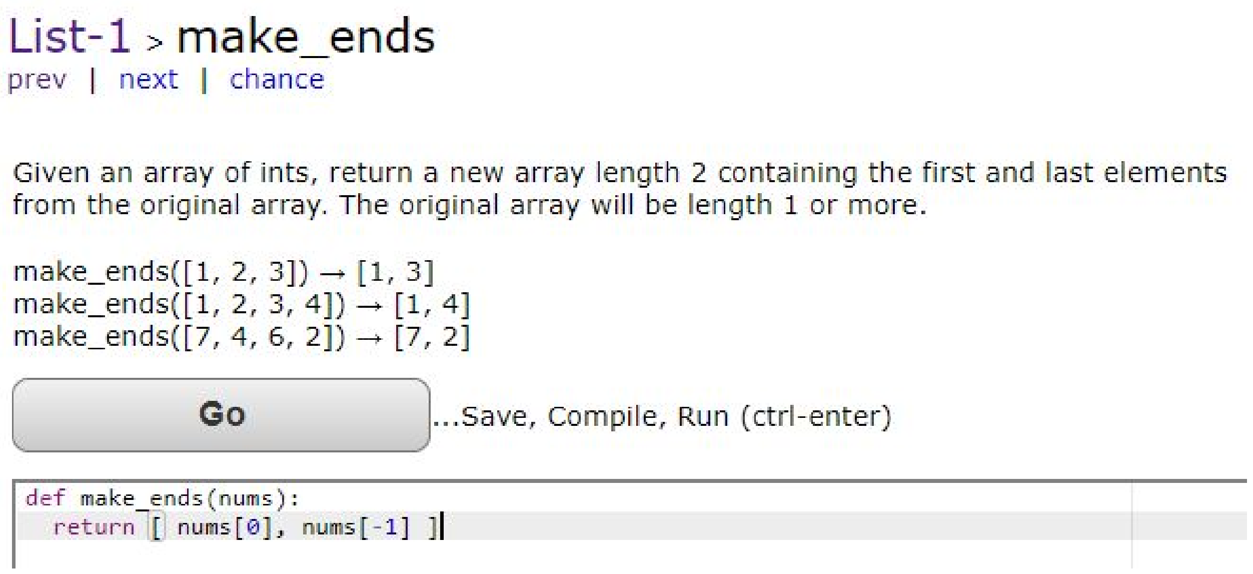
List-1 > middle_way

prev | next | chance

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]

```python
def middle_way(a, b):
  return [ a[1], b[1] ]
```



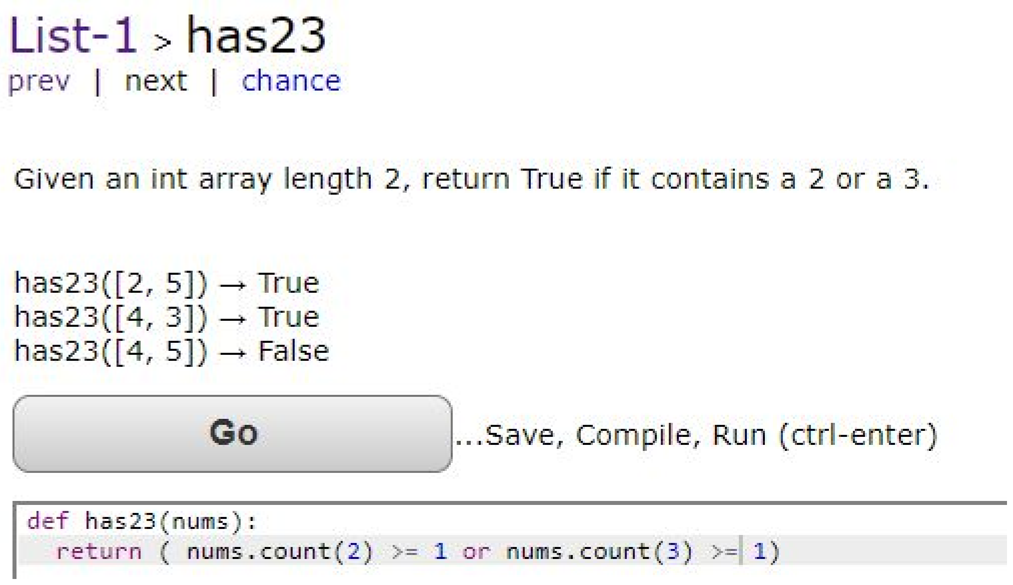
List-1 > make_ends

prev | next | chance

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]

```python
def make_ends(nums):
  return [ nums[0], nums[-1] ]
```



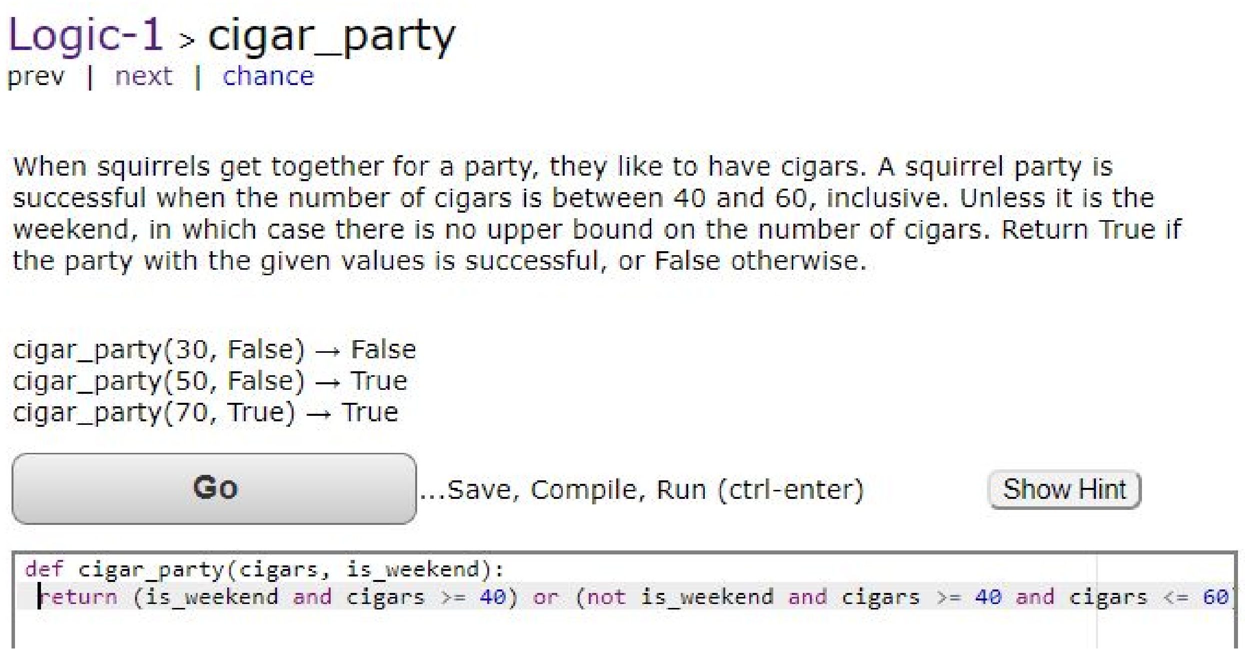
List-1 > has23

prev | next | chance

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

```python
def has23(nums):
  return ( nums.count(2) >= 1 or nums.count(3) >= 1)
```



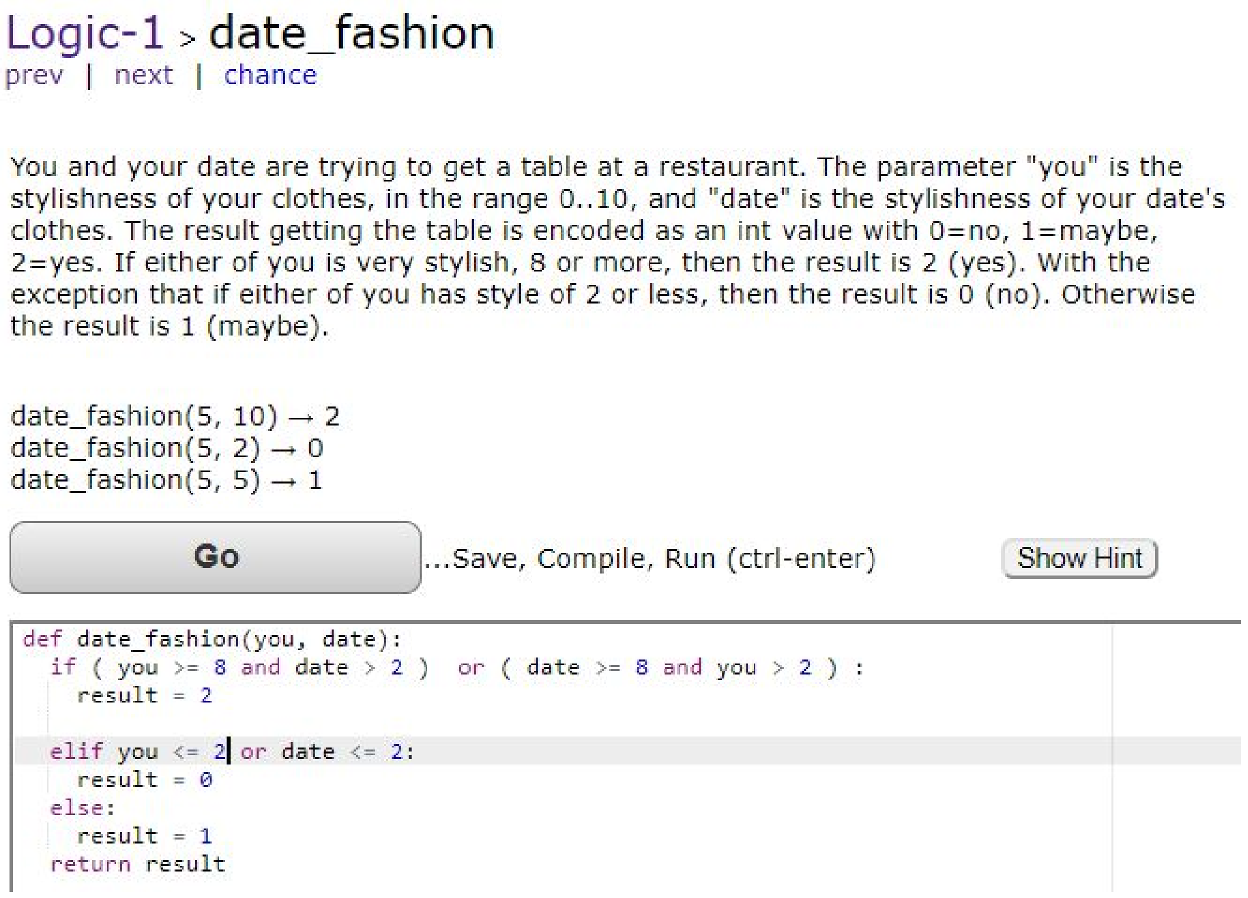
Logic-1 > cigar_party

prev | next | chance

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 (is_weekend and cigars >= 40) or (not is_weekend and cigars >= 40 and cigars <= 60)
```



Logic-1 > date_fashion

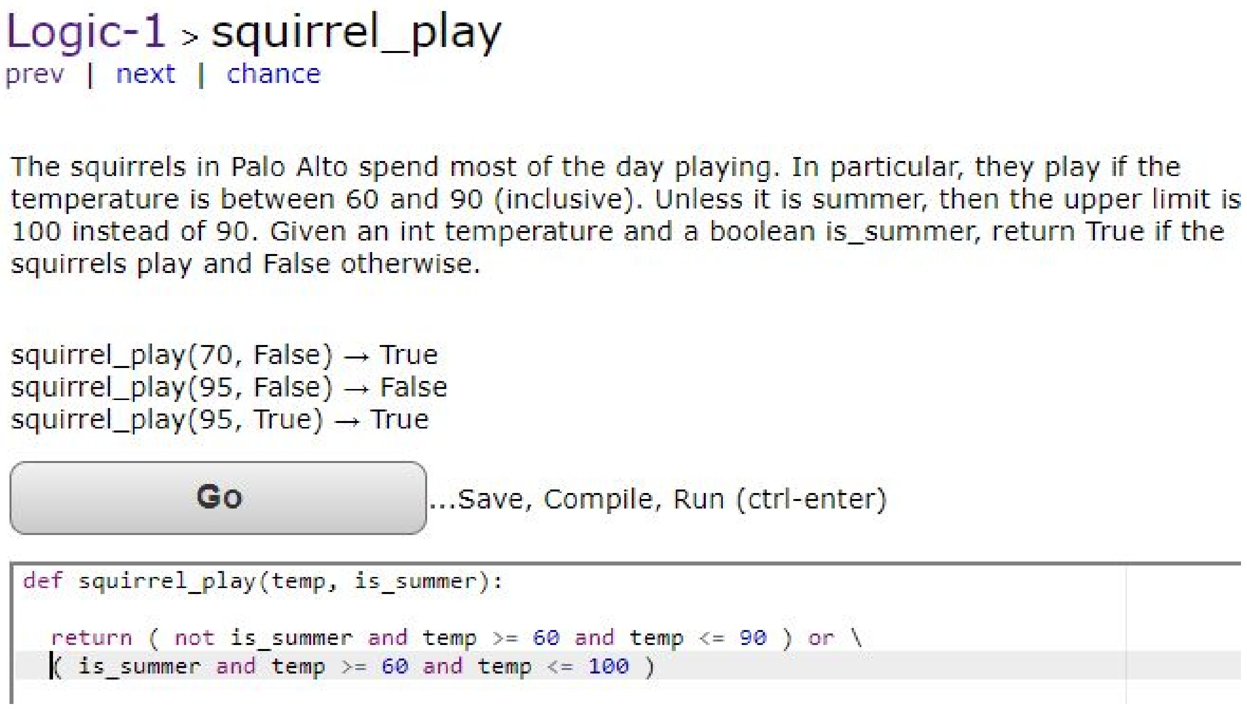
prev | next | chance

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 >= 8 and date > 2 )  or ( date >= 8 and you > 2 ) :
    result = 2

  elif you <= 2 or date <= 2:
    result = 0
  else:
    result = 1
  return result
```



Logic-1 > squirrel_play

prev | next | chance

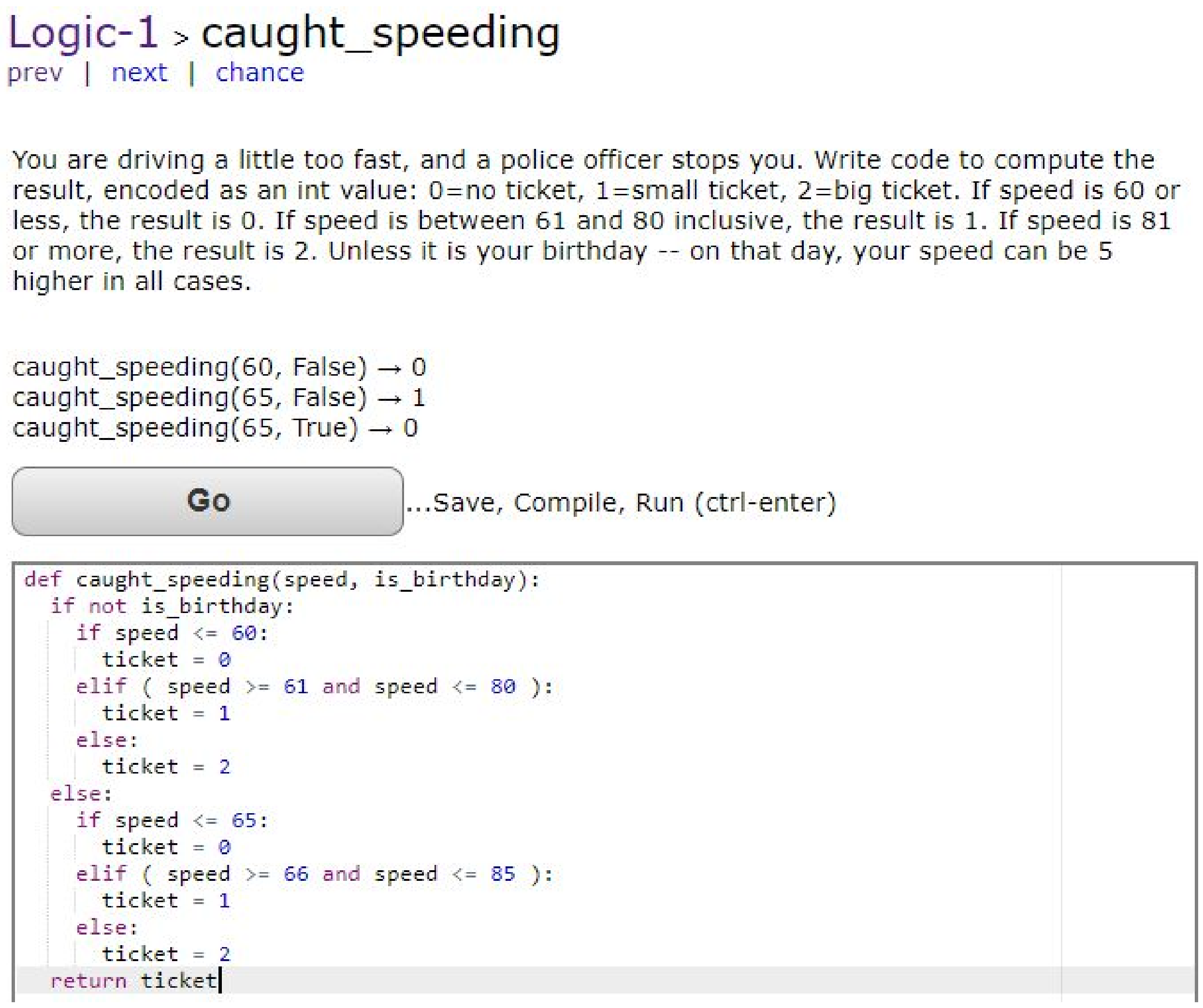
The squirrels in Palo Alto spend most of the day playing. In particular, they play if the temperature is between 60 and 90 (inclusive). Unless it is summer, then the upper limit is 100 instead of 90. Given an int temperature and a boolean is_summer, return True if the squirrels play and False otherwise.

squirrel_play(70, False) → True
squirrel_play(95, False) → False
squirrel_play(95, True) → True

Go ...Save, Compile, Run (ctrl-enter)

```
def squirrel_play(temp, is_summer):

  return ( not is_summer and temp >= 60 and temp <= 90 ) or \
  ( is_summer and temp >= 60 and temp <= 100 )
```



Logic-1 > caught_speeding

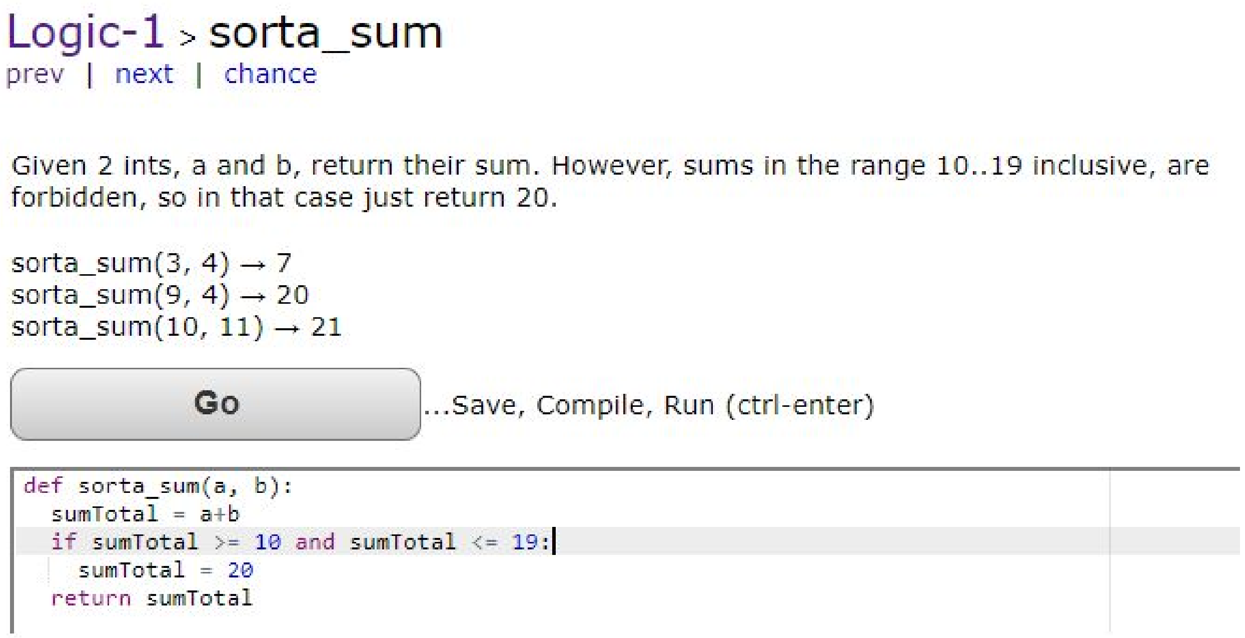
prev | next | chance

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

Go ...Save, Compile, Run (ctrl-enter)

```python
def caught_speeding(speed, is_birthday):
  if not is_birthday:
    if speed <= 60:
      ticket = 0
    elif ( speed >= 61 and speed <= 80 ):
      ticket = 1
    else:
      ticket = 2
  else:
    if speed <= 65:
      ticket = 0
    elif ( speed >= 66 and speed <= 85 ):
      ticket = 1
    else:
      ticket = 2
  return ticket
```



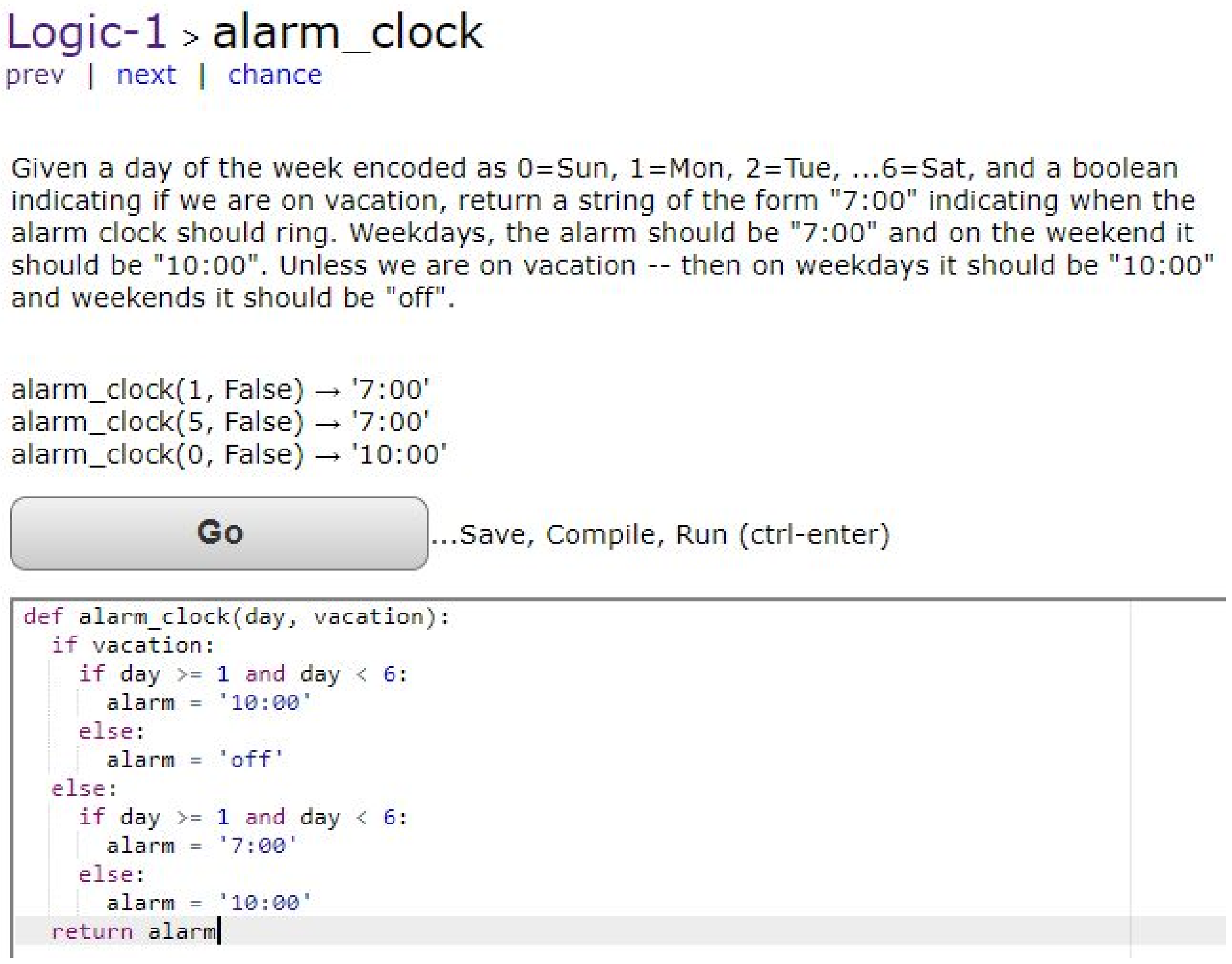
Logic-1 > sorta_sum

prev | next | chance

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

```python
def sorta_sum(a, b):
  sumTotal = a+b
  if sumTotal >= 10 and sumTotal <= 19:
    sumTotal = 20
  return sumTotal
```


Logic-1 > alarm_clock

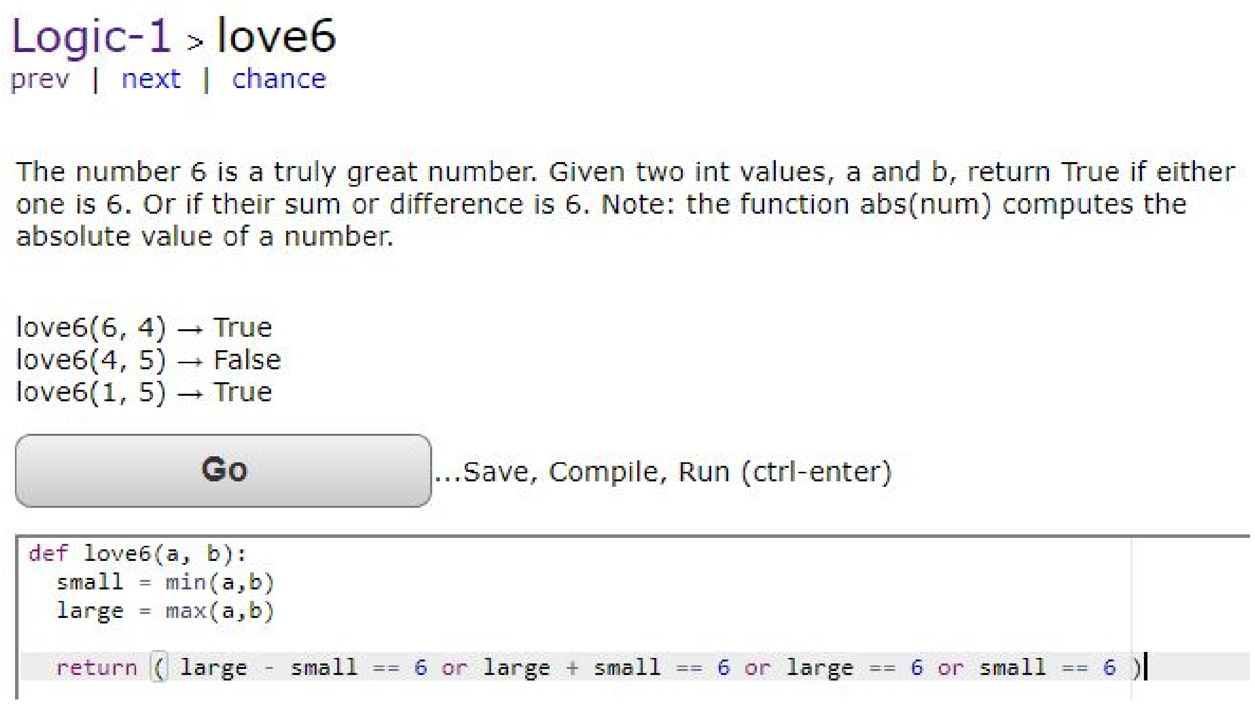
prev | next | chance

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'

Go ...Save, Compile, Run (ctrl-enter)

```
def alarm_clock(day, vacation):
  if vacation:
    if day >= 1 and day < 6:
      alarm = '10:00'
    else:
      alarm = 'off'
  else:
    if day >= 1 and day < 6:
      alarm = '7:00'
    else:
      alarm = '10:00'
  return alarm
```



Logic-1 > love6

prev | next | chance

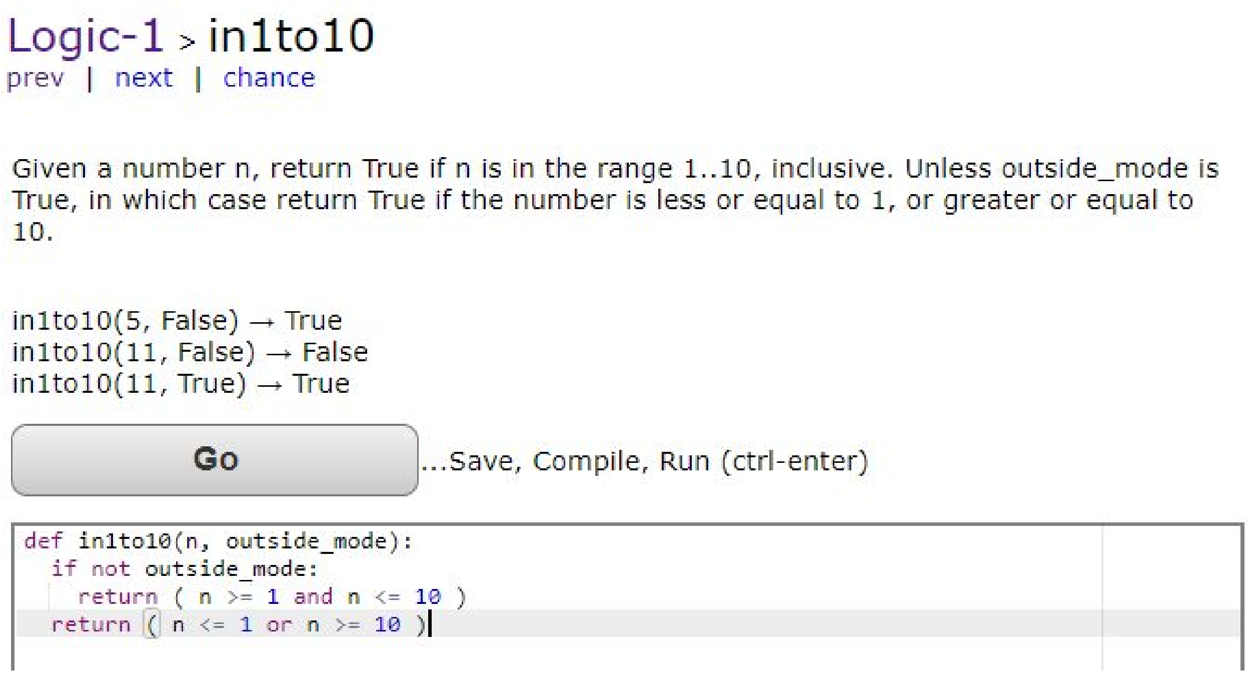
The number 6 is a truly great number. Given two int values, a and b, return True if either one is 6. Or if their sum or difference is 6. Note: the function abs(num) computes the absolute value of a number.

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

Go ...Save, Compile, Run (ctrl-enter)

```
def love6(a, b):
  small = min(a,b)
  large = max(a,b)

  return ( large - small == 6 or large + small == 6 or large == 6 or small == 6 )
```



Logic-1 > in1to10

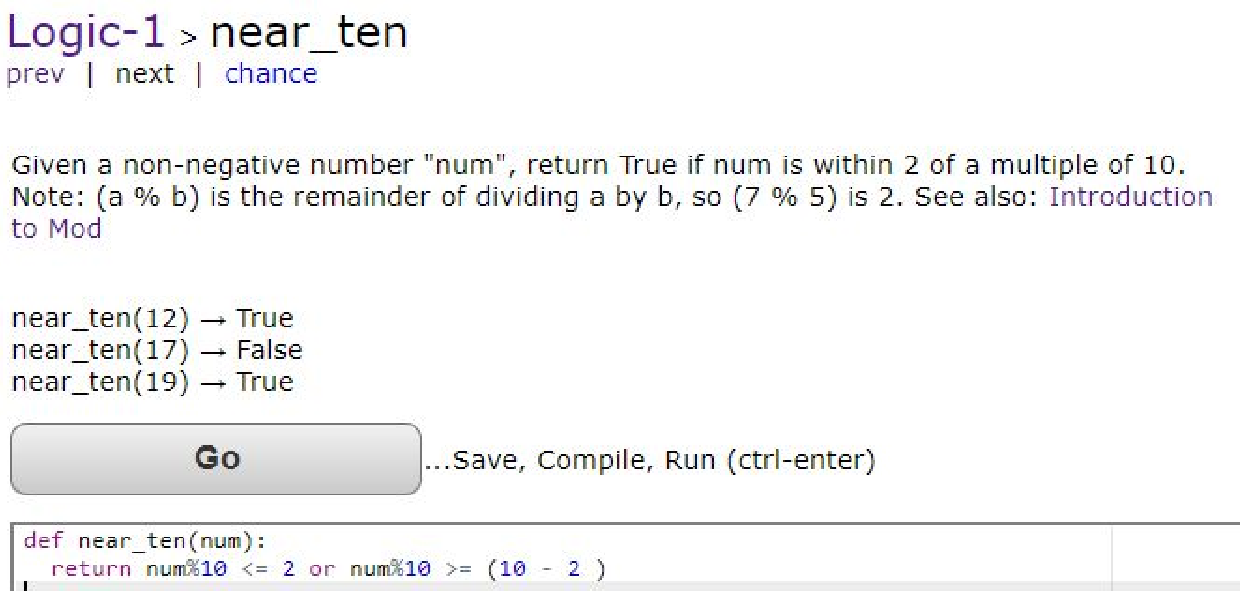
prev | next | chance

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

Go ...Save, Compile, Run (ctrl-enter)

```
def in1to10(n, outside_mode):
  if not outside_mode:
    return ( n >= 1 and n <= 10 )
  return ( n <= 1 or n >= 10 )
```



Logic-1 > near_ten

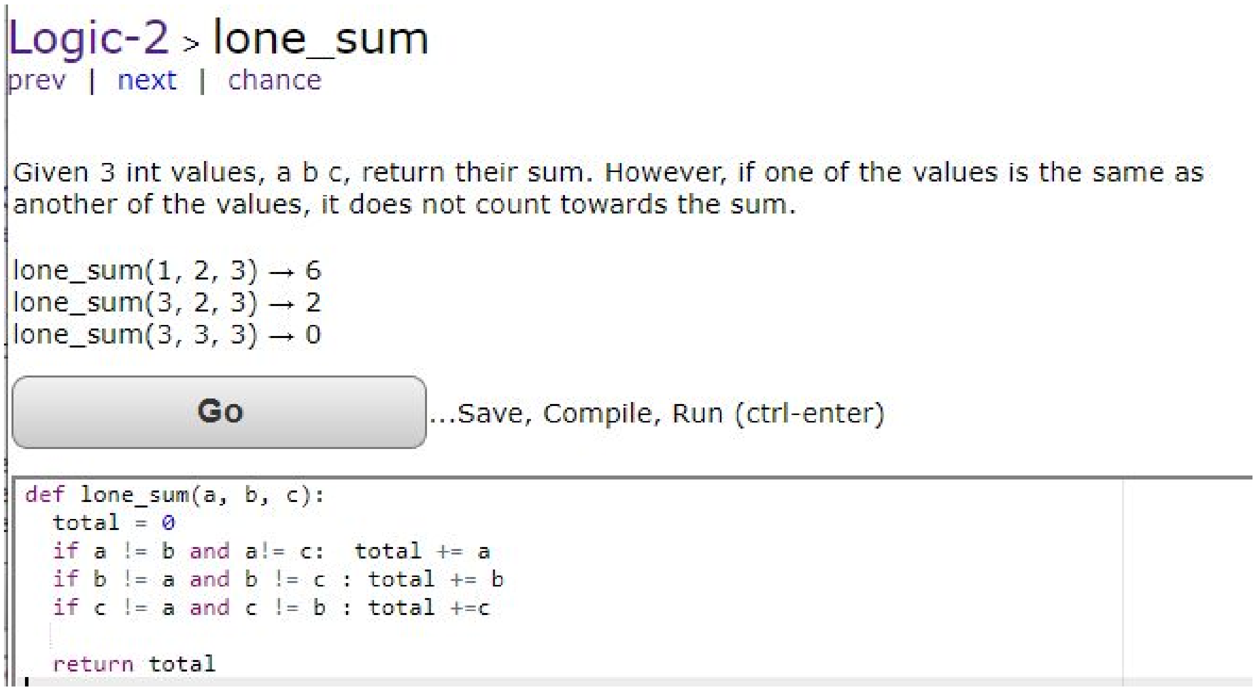
prev | next | chance

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

near_ten(12) → True
near_ten(17) → False
near_ten(19) → True

Go ...Save, Compile, Run (ctrl-enter)

```
def near_ten(num):
  return num%10 <= 2 or num%10 >= (10 - 2 )
```



Logic-2 > lone_sum

prev | next | chance

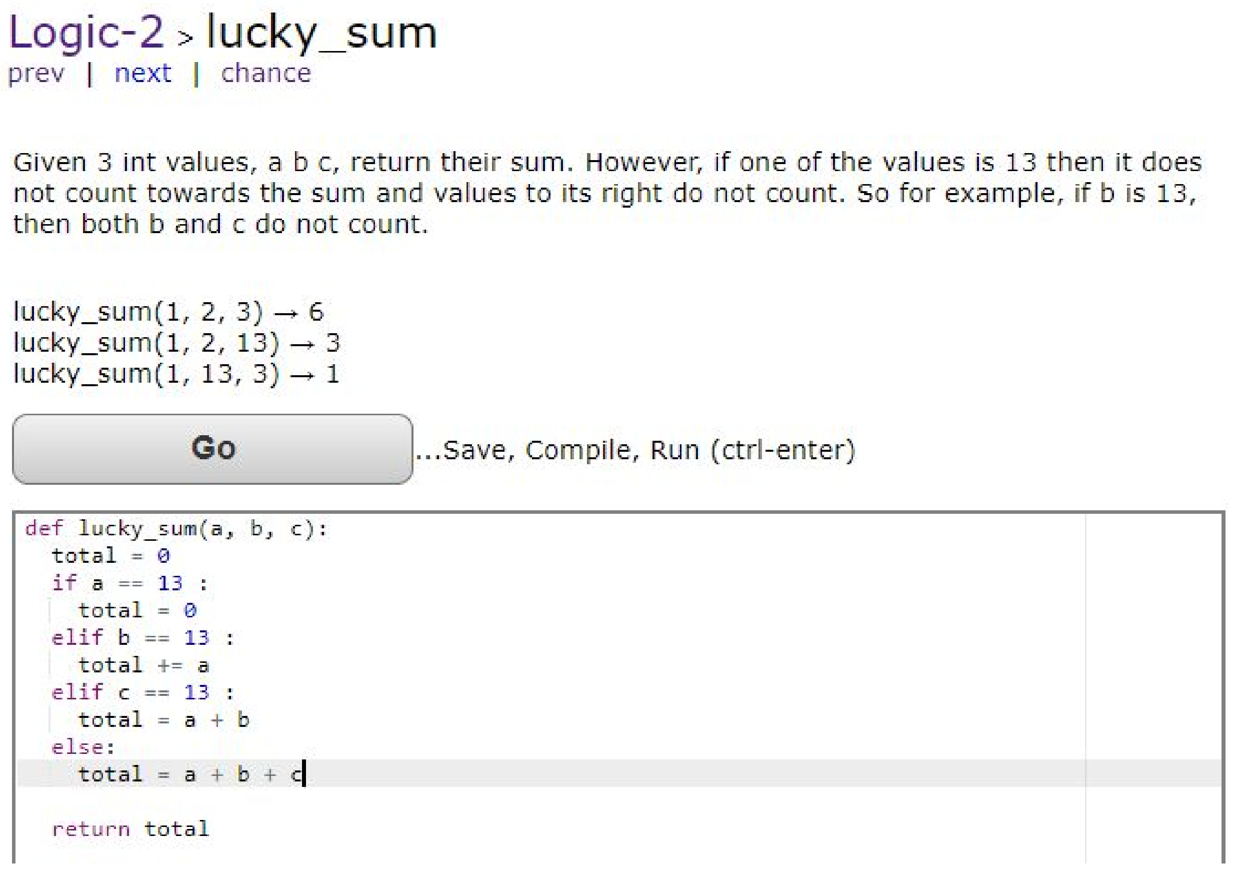
Given 3 int values, a b c, return their sum. However, if one of the values is the same as another of the values, it does not count towards the sum.

lone_sum(1, 2, 3) → 6
lone_sum(3, 2, 3) → 2
lone_sum(3, 3, 3) → 0

Go ...Save, Compile, Run (ctrl-enter)

```
def lone_sum(a, b, c):
  total = 0
  if a != b and a!= c:  total += a
  if b != a and b != c : total += b
  if c != a and c != b : total +=c

  return total
```



Logic-2 > lucky_sum

prev | next | chance

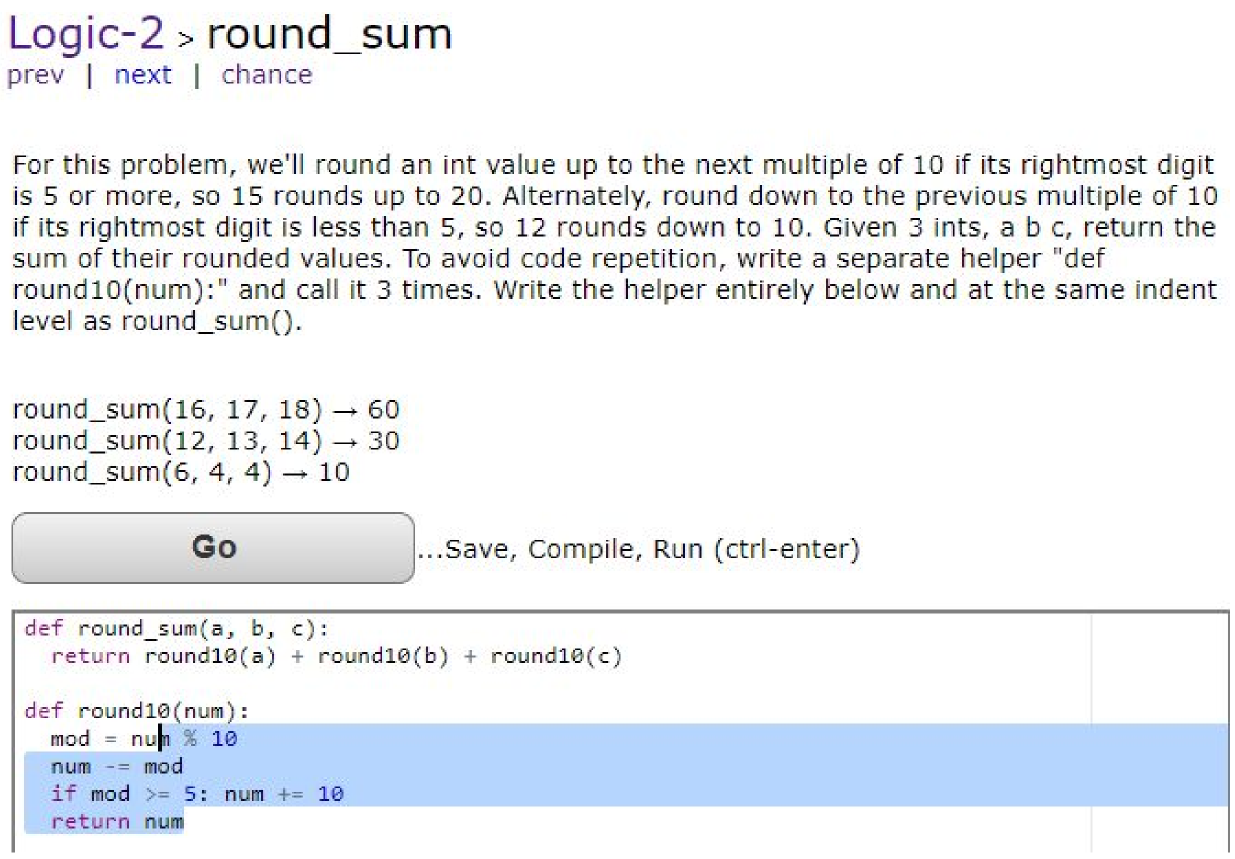
Given 3 int values, a b c, return their sum. However, if one of the values is 13 then it does not count towards the sum and values to its right do not count. So for example, if b is 13, then both b and c do not count.

lucky_sum(1, 2, 3) → 6
lucky_sum(1, 2, 13) → 3
lucky_sum(1, 13, 3) → 1

Go ...Save, Compile, Run (ctrl-enter)

```python
def lucky_sum(a, b, c):
  total = 0
  if a == 13 :
    total = 0
  elif b == 13 :
    total += a
  elif c == 13 :
    total = a + b
  else:
    total = a + b + c

  return total
```



Logic-2 > round_sum

prev | next | chance

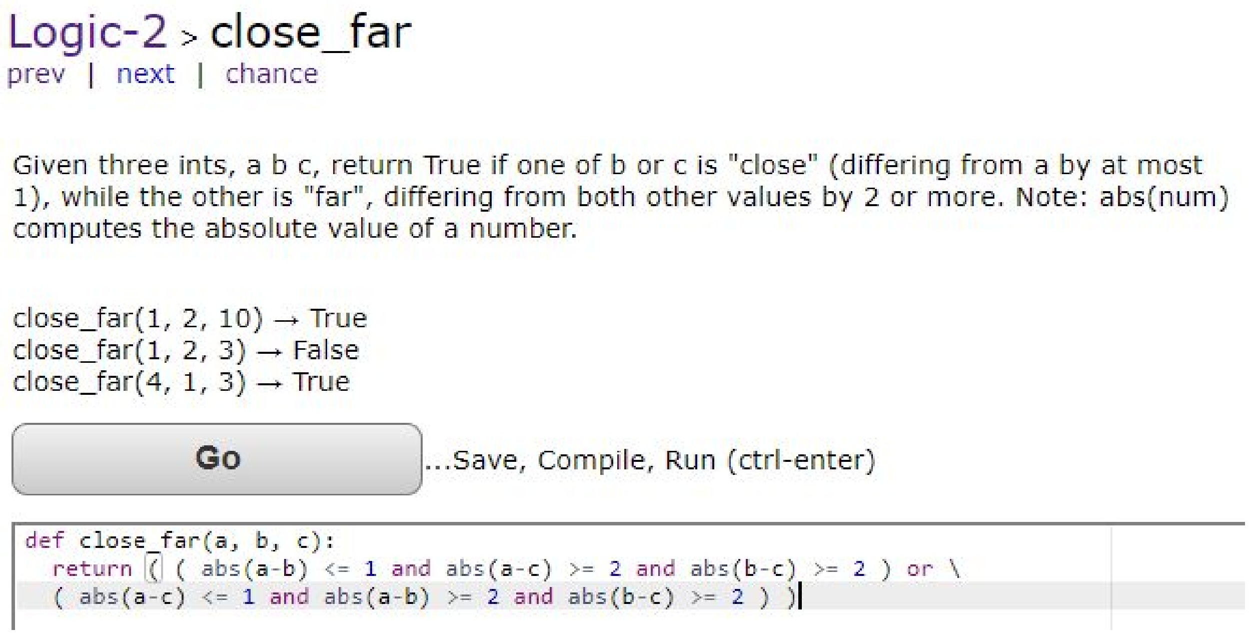
For this problem, we'll round an int value up to the next multiple of 10 if its rightmost digit is 5 or more, so 15 rounds up to 20. Alternately, round down to the previous multiple of 10 if its rightmost digit is less than 5, so 12 rounds down to 10. Given 3 ints, a b c, return the sum of their rounded 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

Go ...Save, Compile, Run (ctrl-enter)

```
def round_sum(a, b, c):
  return round10(a) + round10(b) + round10(c)

def round10(num):
  mod = num % 10
  num -= mod
  if mod >= 5: num += 10
  return num
```



Logic-2 > close_far

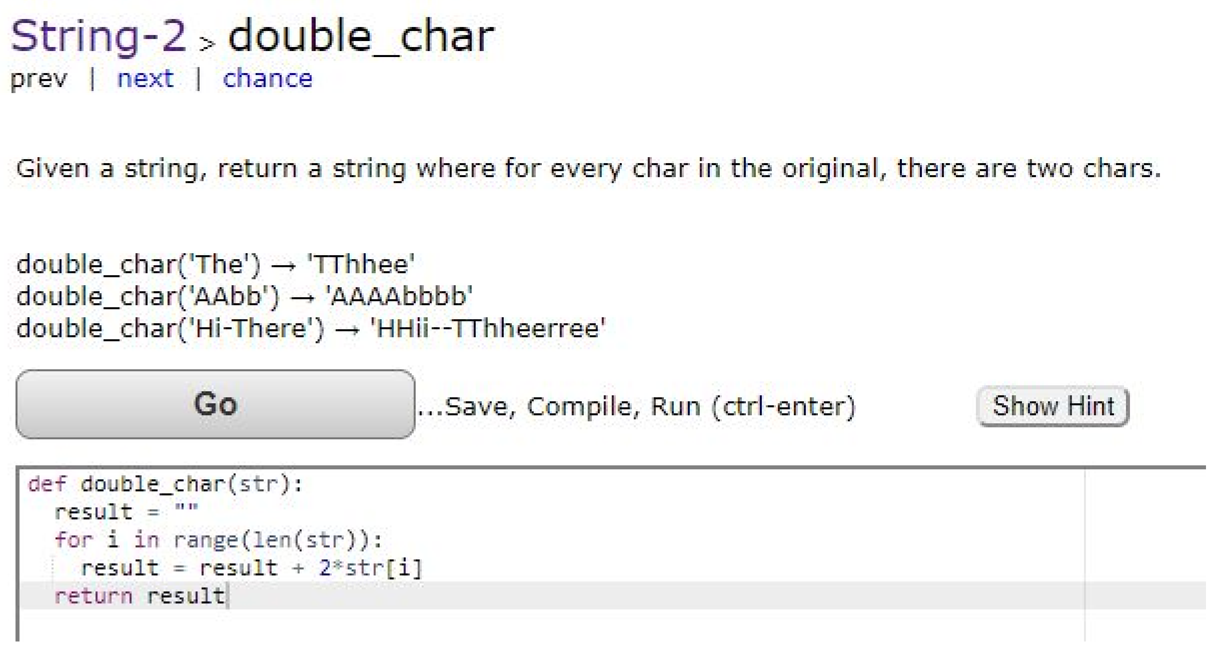
prev | next | chance

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

Go ...Save, Compile, Run (ctrl-enter)

```
def close_far(a, b, c):
  return ( ( abs(a-b) <= 1 and abs(a-c) >= 2 and abs(b-c) >= 2 ) or \
  ( abs(a-c) <= 1 and abs(a-b) >= 2 and abs(b-c) >= 2 ) )
```



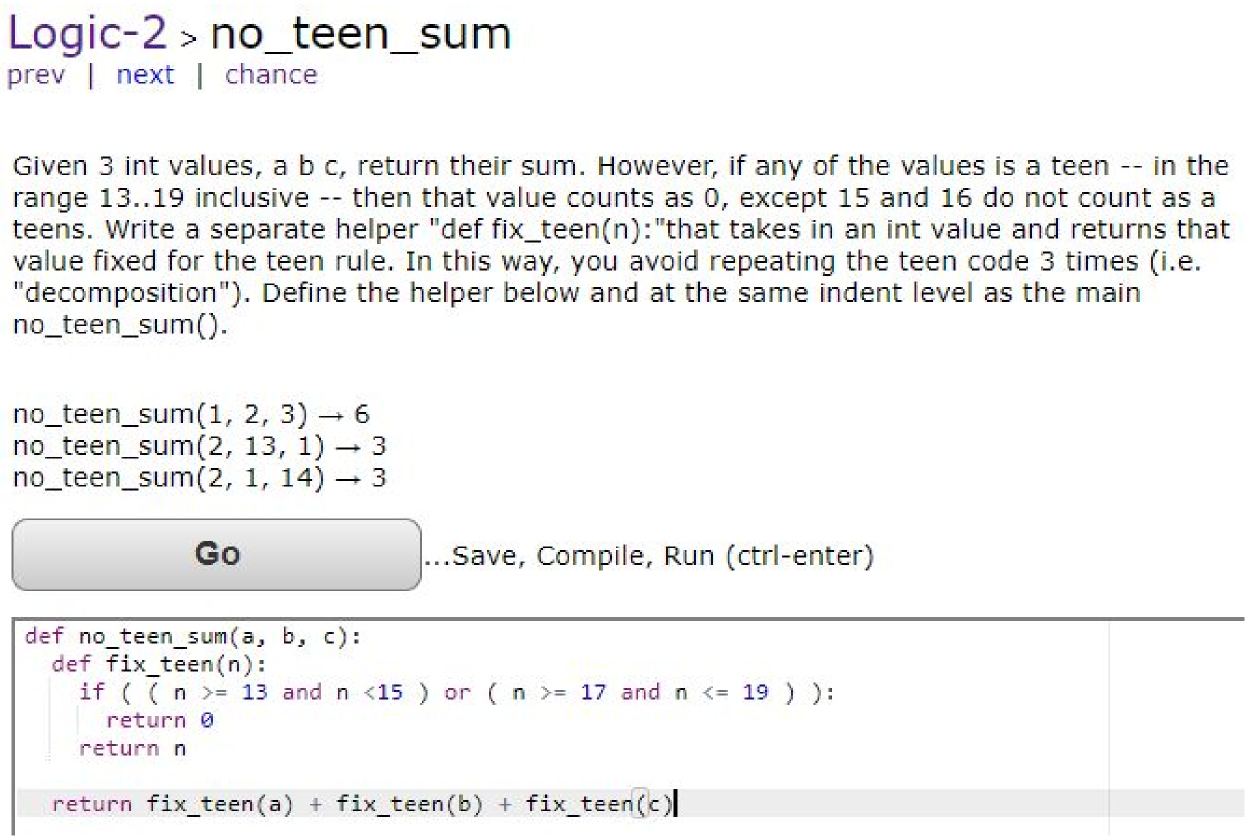
String-2 > double_char

prev | next | chance

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):
  result = ""
  for i in range(len(str)):
    result = result + 2*str[i]
  return result
```



Logic-2 > no_teen_sum

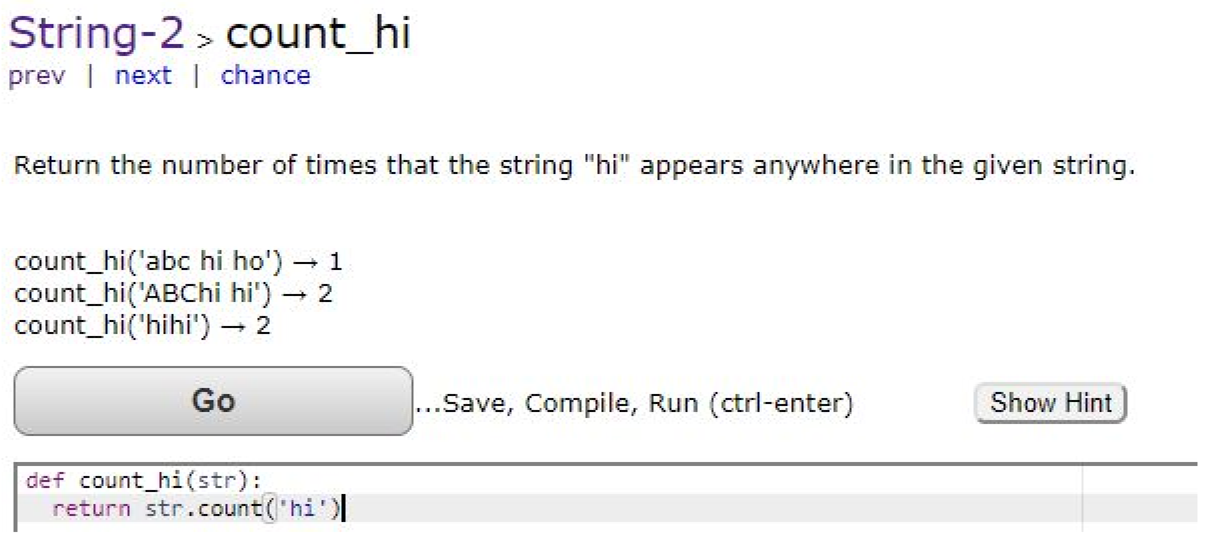
prev | next | chance

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):
  def fix_teen(n):
    if ( ( n >= 13 and n <15 ) or ( n >= 17 and n <= 19 ) ):
      return 0
    return n

  return fix_teen(a) + fix_teen(b) + fix_teen(c)
```



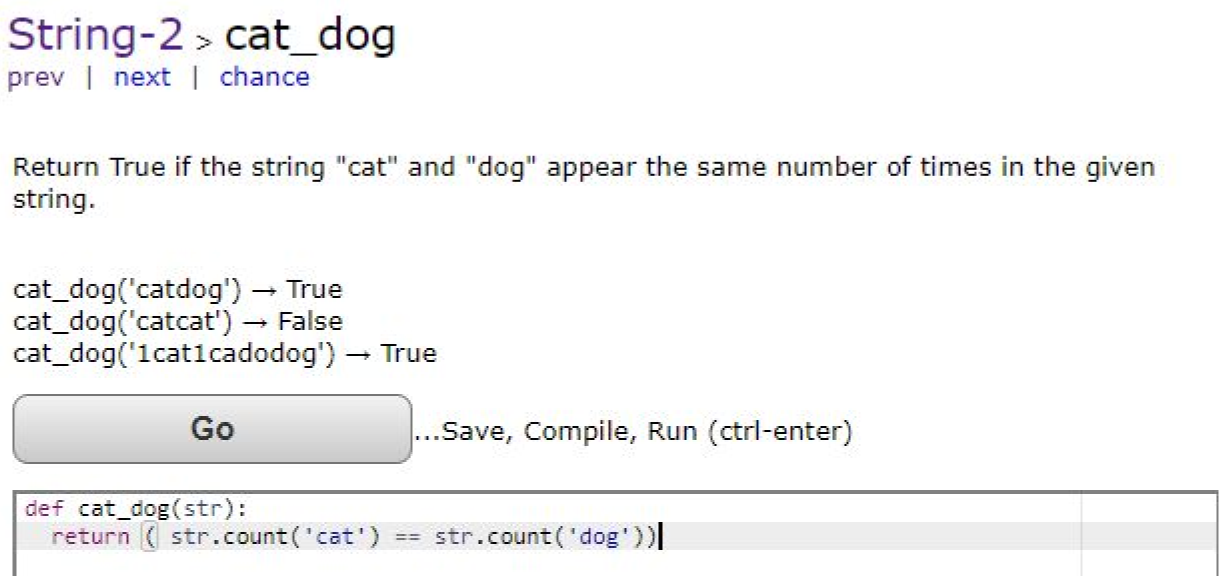
String-2 > count_hi

prev | next | chance

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):
  return str.count('hi')
```



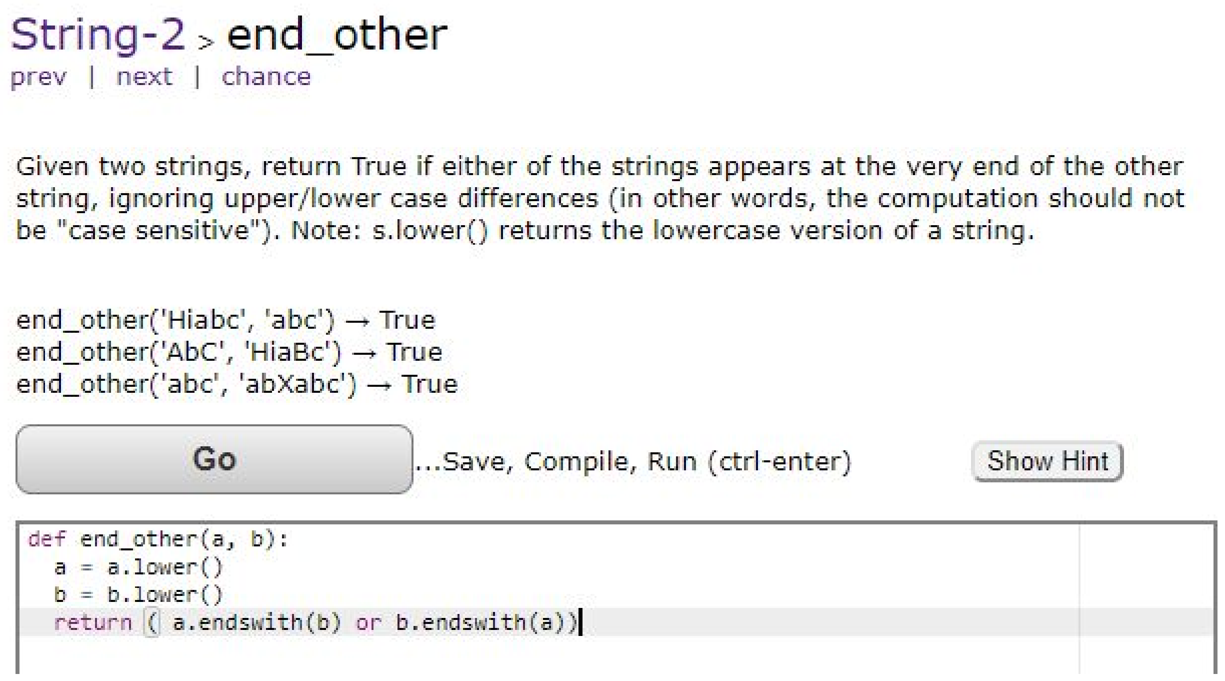
String-2 > cat_dog

prev | next | chance

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):
  return ( str.count('cat') == str.count('dog'))
```



String-2 > end_other

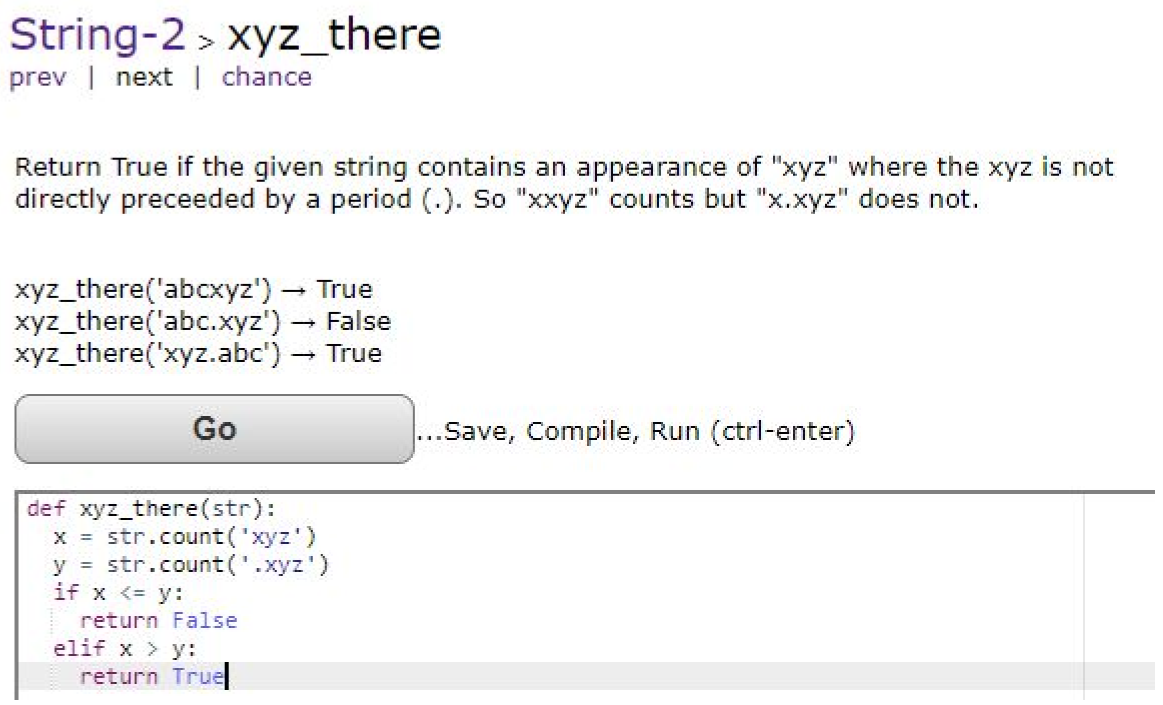
prev | next | chance

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

...Save, Compile, Run (ctrl-enter)

```
def end_other(a, b):
  a = a.lower()
  b = b.lower()
  return ( a.endswith(b) or b.endswith(a))
```



String-2 > xyz_there

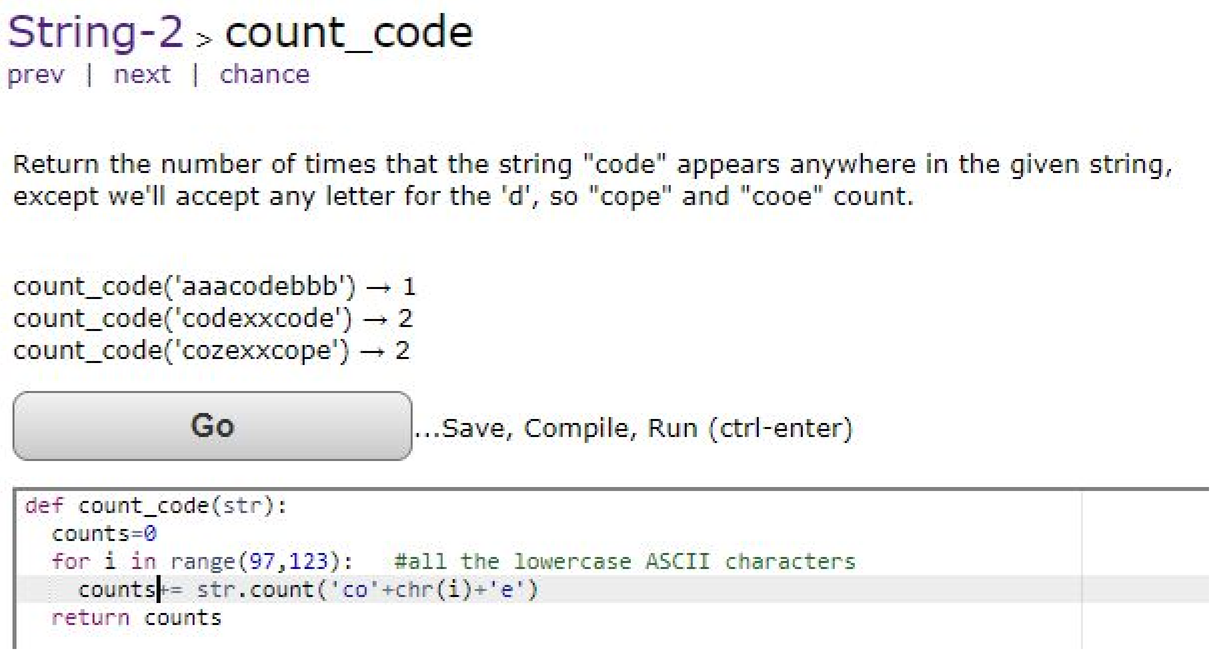
prev | next | chance

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

...Save, Compile, Run (ctrl-enter)

```
def xyz_there(str):
  x = str.count('xyz')
  y = str.count('.xyz')
  if x <= y:
    return False
  elif x > y:
    return True
```



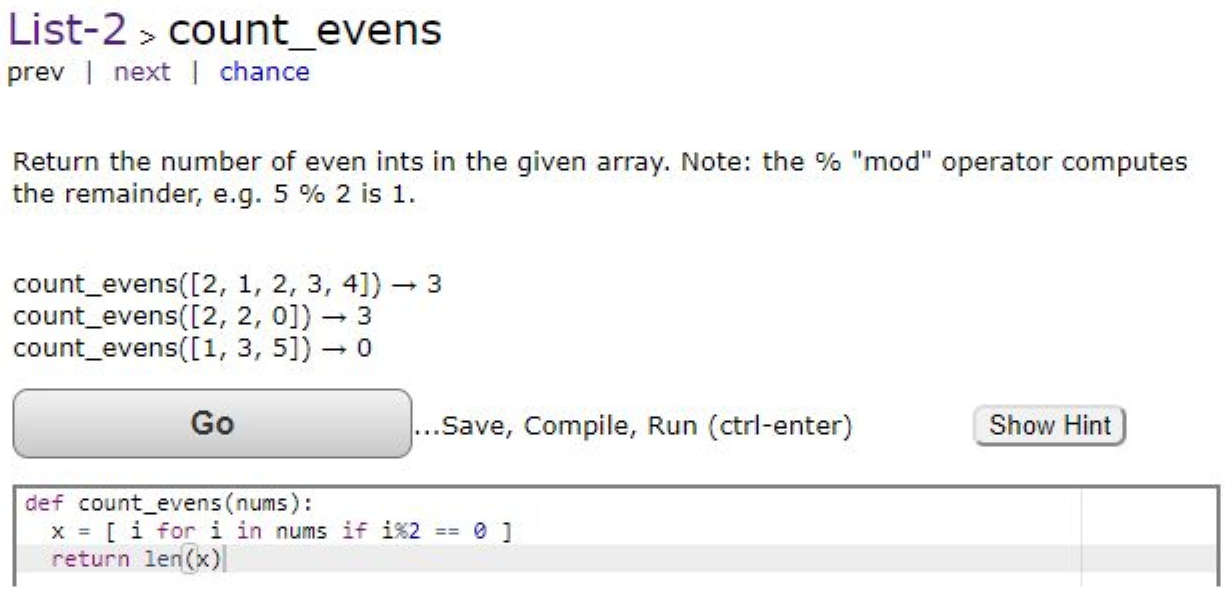
String-2 > count_code

prev | next | chance

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):
  counts=0
  for i in range(97,123):   #all the lowercase ASCII characters
    counts+= str.count('co'+chr(i)+'e')
  return counts
```



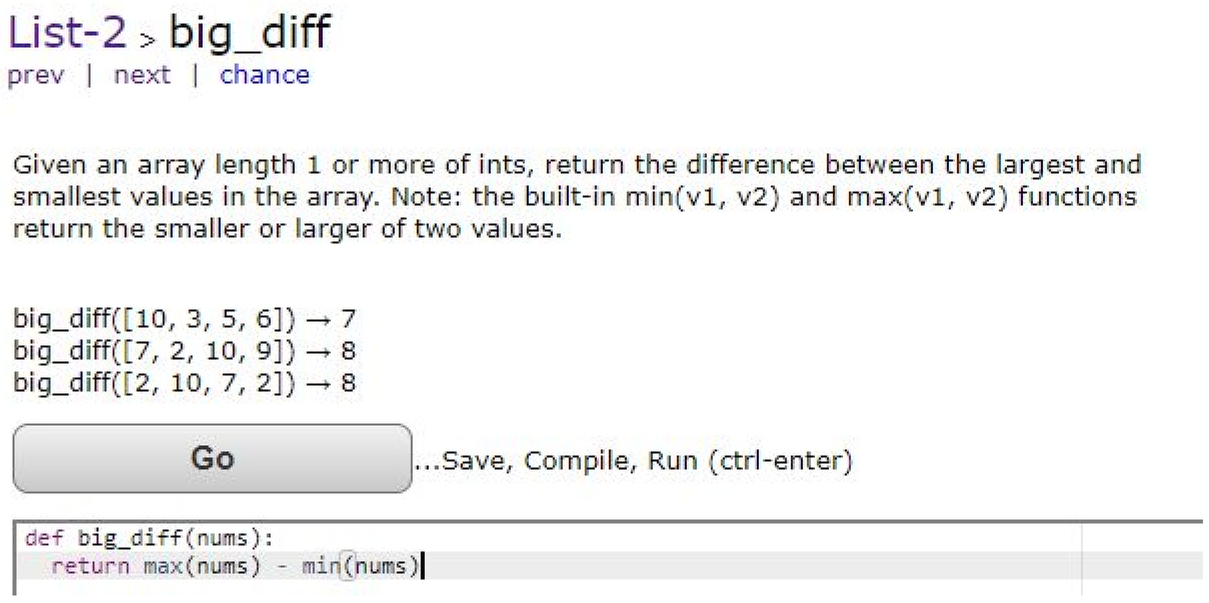
List-2 > count_evens

prev | next | chance

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):
  x = [ i for i in nums if i%2 == 0 ]
  return len(x)
```



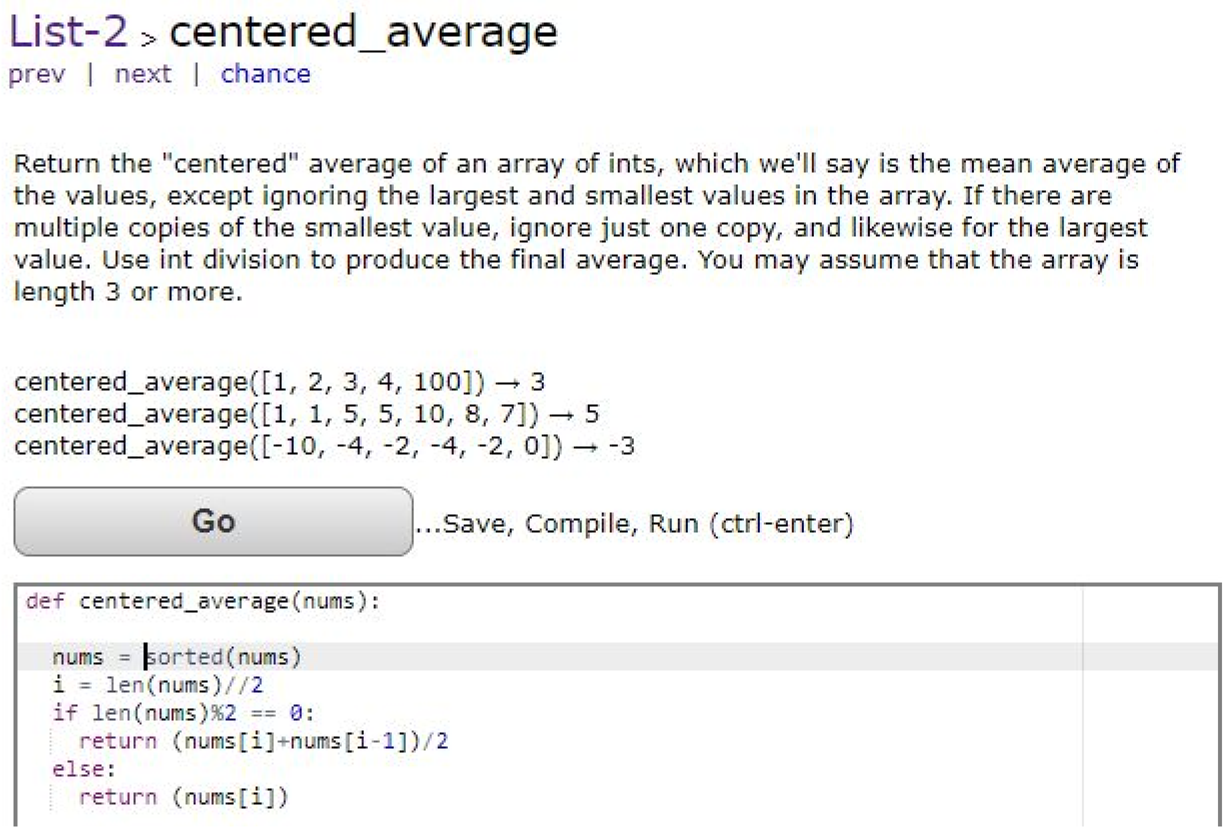
List-2 > big_diff

prev | next | chance

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)
```



List-2 > centered_average

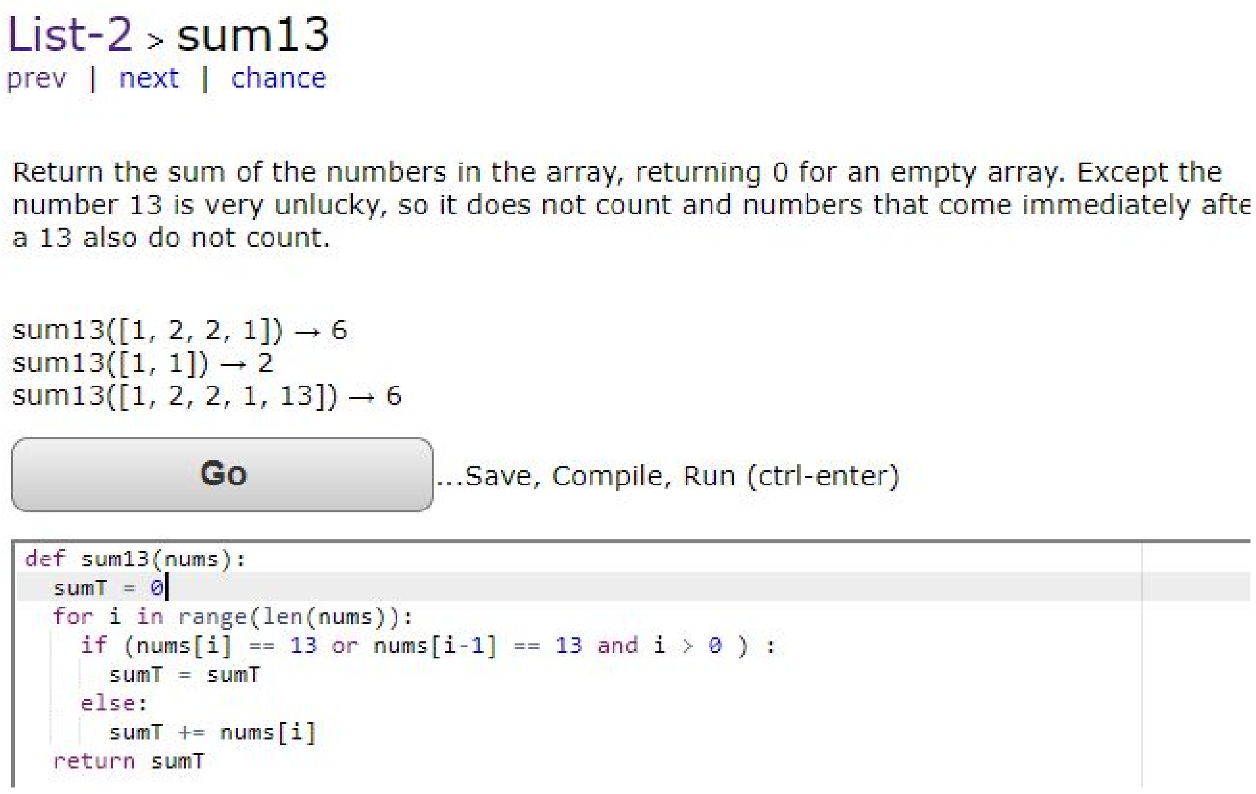
prev | next | chance

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):

  nums = sorted(nums)
  i = len(nums)//2
  if len(nums)%2 == 0:
    return (nums[i]+nums[i-1])/2
  else:
    return (nums[i])
```



List-2 > sum13

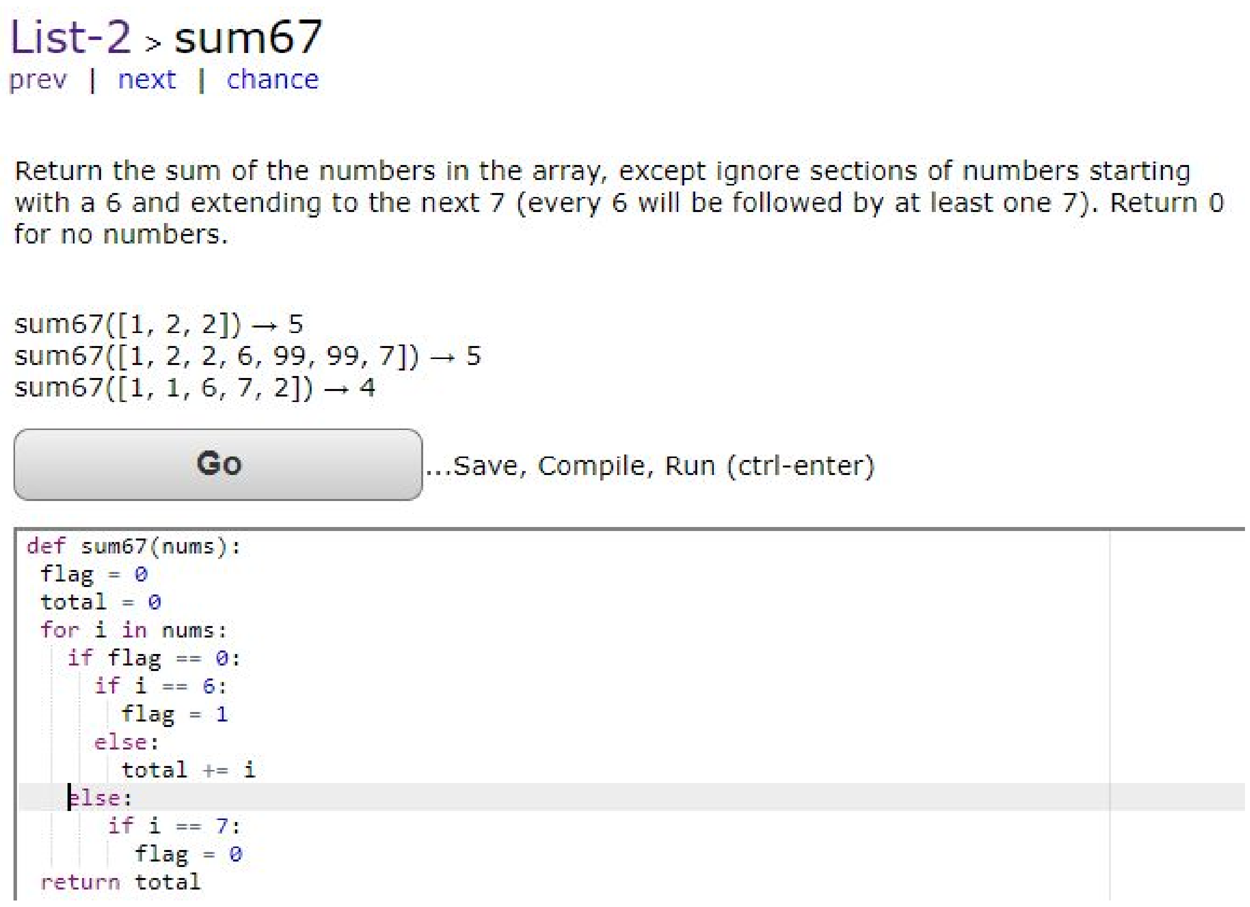
prev | next | chance

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 afte a 13 also do not count.

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

Go ...Save, Compile, Run (ctrl-enter)

```
def sum13(nums):
  sumT = 0
  for i in range(len(nums)):
    if (nums[i] == 13 or nums[i-1] == 13 and i > 0 ) :
      sumT = sumT
    else:
      sumT += nums[i]
  return sumT
```

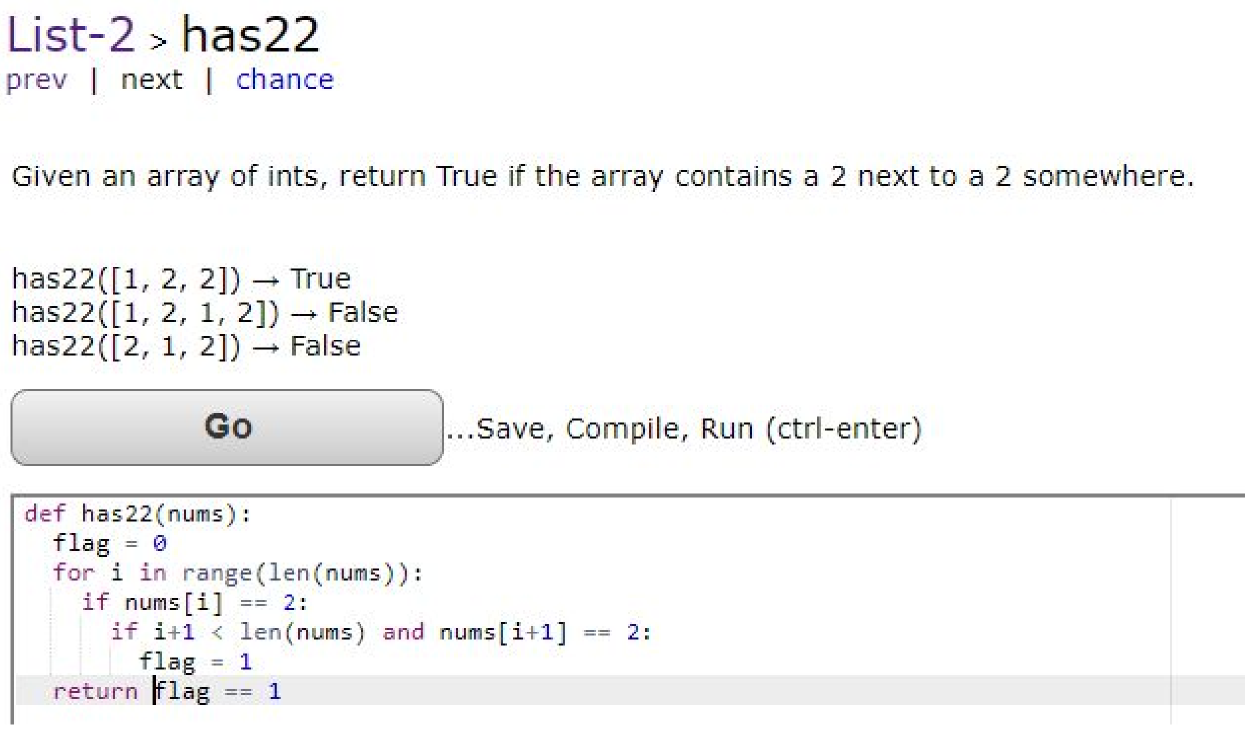


List-2 > sum67
prev | next | chance

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

```python
def sum67(nums):
 flag = 0
 total = 0
 for i in nums:
   if flag == 0:
     if i == 6:
       flag = 1
     else:
       total += i
   else:
      if i == 7:
        flag = 0
 return total
```

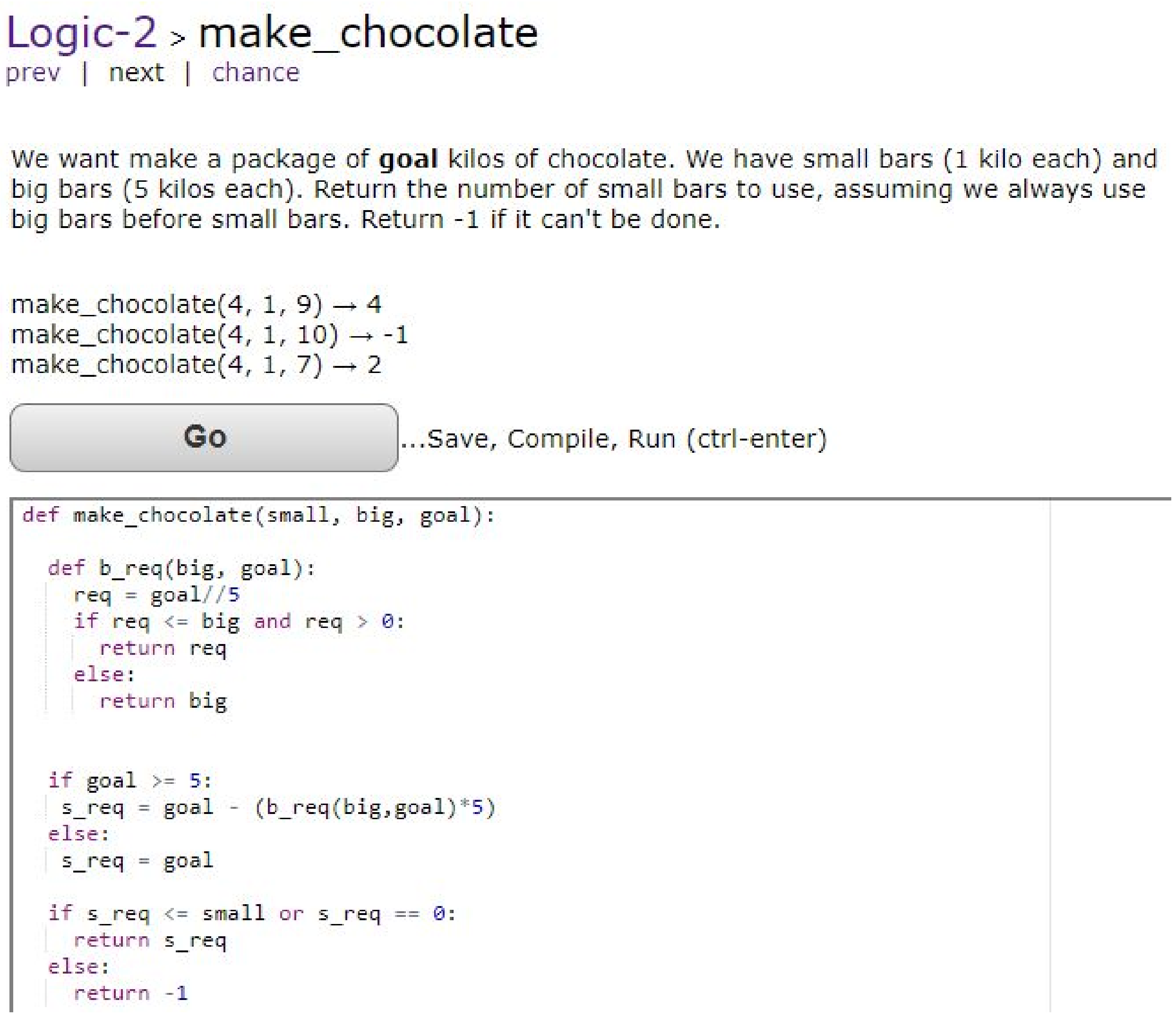


List-2 > has22
prev | next | chance

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

```python
def has22(nums):
  flag = 0
  for i in range(len(nums)):
    if nums[i] == 2:
      if i+1 < len(nums) and nums[i+1] == 2:
        flag = 1
  return flag == 1
```



Logic-2 > make_chocolate

prev | next | chance

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

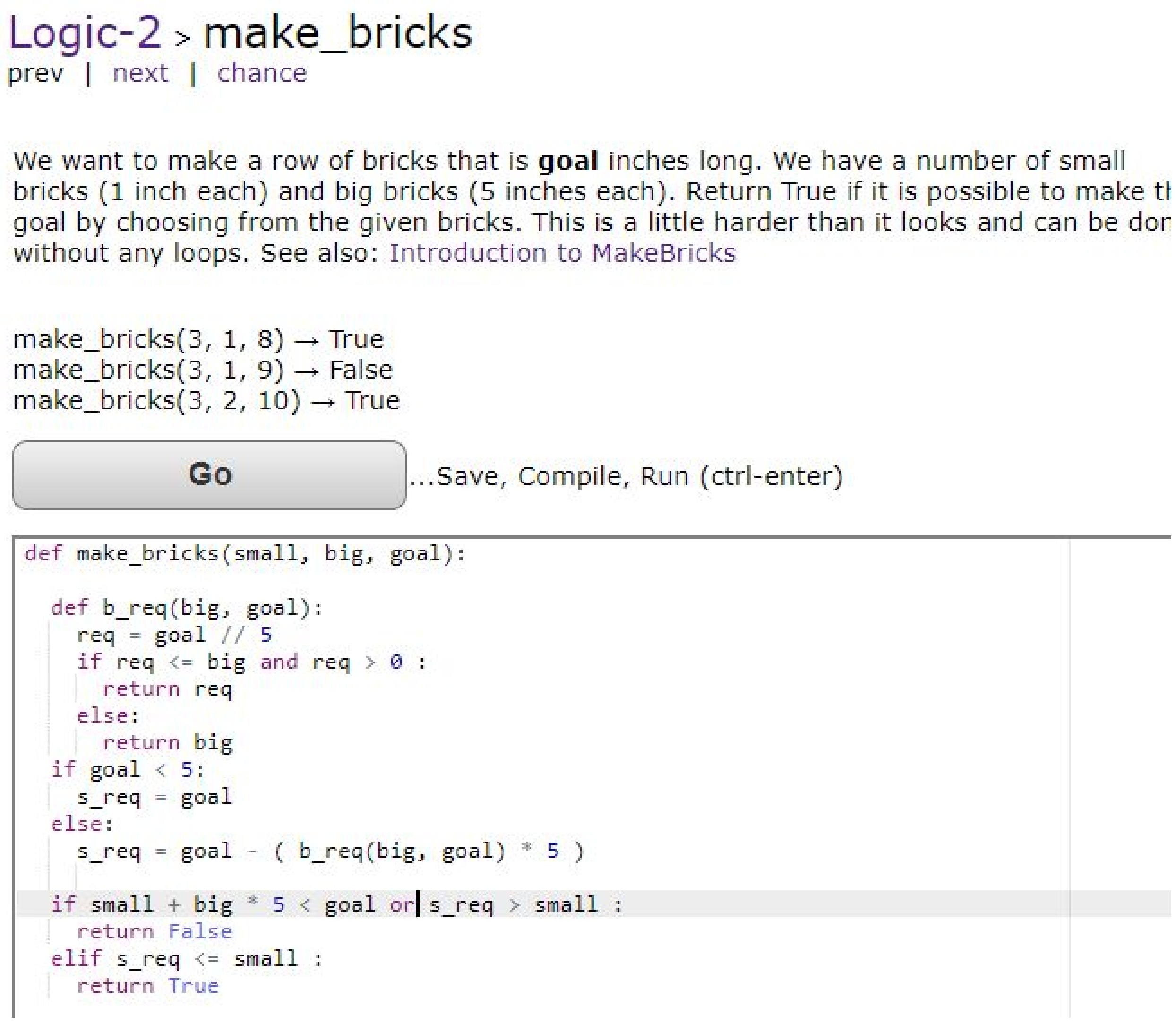
...Save, Compile, Run (ctrl-enter)

```
def make_chocolate(small, big, goal):

  def b_req(big, goal):
    req = goal//5
    if req <= big and req > 0:
      return req
    else:
      return big


  if goal >= 5:
   s_req = goal - (b_req(big,goal)*5)
  else:
   s_req = goal

  if s_req <= small or s_req == 0:
    return s_req
  else:
    return -1
```



Logic-2 > make_bricks

prev | next | chance

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 tl goal by choosing from the given bricks. This is a little harder than it looks and can be dor without any loops. See also: Introduction to MakeBricks

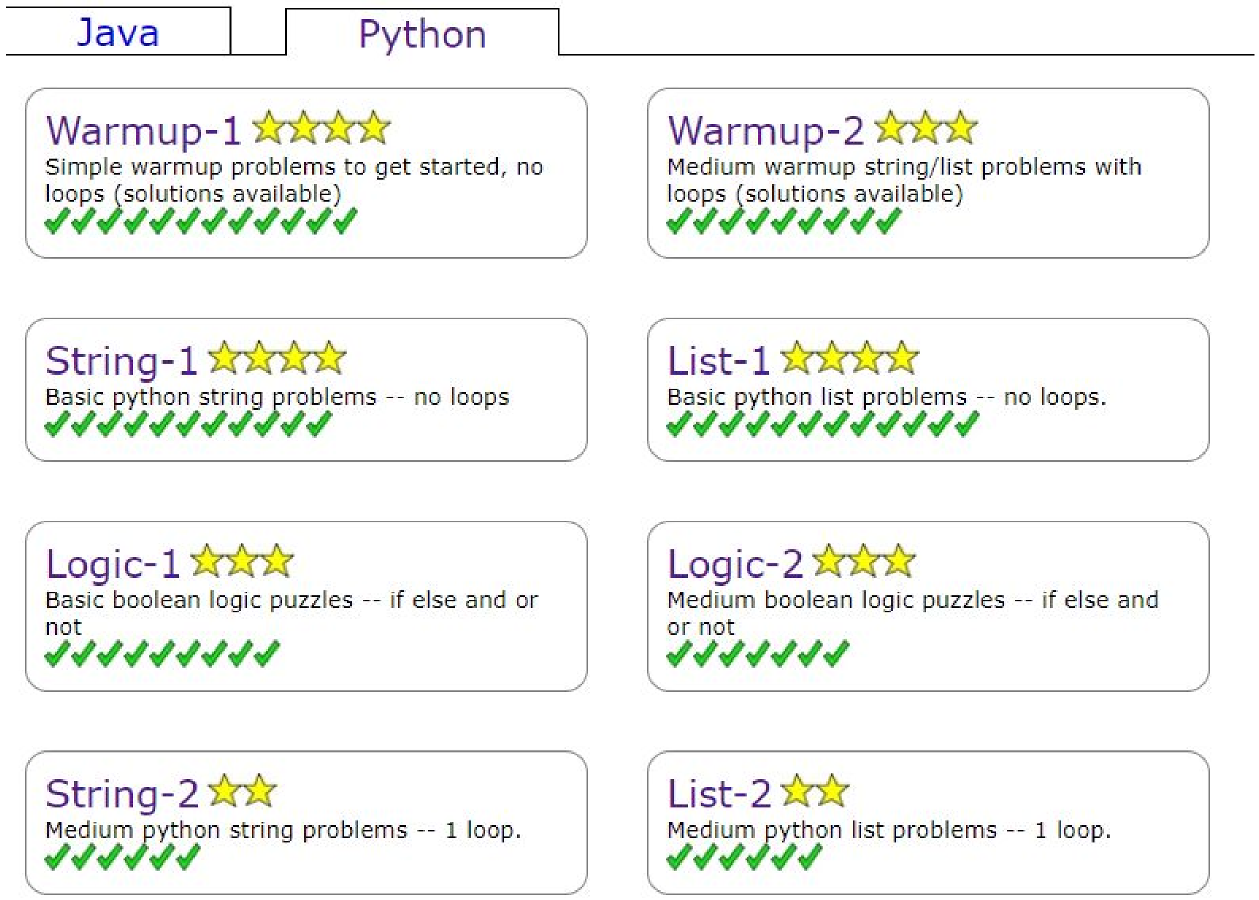
make_bricks(3, 1, 8) → True
make_bricks(3, 1, 9) → False
make_bricks(3, 2, 10) → True

Go ...Save, Compile, Run (ctrl-enter)

```
def make_bricks(small, big, goal):

  def b_req(big, goal):
    req = goal // 5
    if req <= big and req > 0 :
      return req
    else:
      return big
  if goal < 5:
    s_req = goal
  else:
    s_req = goal - ( b_req(big, goal) * 5 )

  if small + big * 5 < goal or s_req > small :
    return False
  elif s_req <= small :
    return True
```



Java | Python

Warmup-1 ★★★★
Simple warmup problems to get started, no loops (solutions available)
✓✓✓✓✓✓✓✓✓✓✓✓

Warmup-2 ★★★
Medium warmup string/list problems with loops (solutions available)
✓✓✓✓✓✓✓✓✓

String-1 ★★★★
Basic python string problems -- no loops
✓✓✓✓✓✓✓✓✓✓✓

List-1 ★★★★
Basic python list problems -- no loops.
✓✓✓✓✓✓✓✓✓✓✓✓

Logic-1 ★★★
Basic boolean logic puzzles -- if else and or not
✓✓✓✓✓✓✓✓✓

Logic-2 ★★★
Medium boolean logic puzzles -- if else and or not
✓✓✓✓✓✓✓

String-2 ★★
Medium python string problems -- 1 loop.
✓✓✓✓✓✓

List-2 ★★
Medium python list problems -- 1 loop.
✓✓✓✓✓✓