**STRINGS**

**1) 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+'!'**

**2) 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:]**

**3) 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):**

**l=len(str)**

**needed\_length=l//2**

**char=''**

**for i in range(needed\_length):**

**char=char+str[i]**

**return char**

**4)**

|  |
| --- |
| **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:]+b[1:]**

**5)**

|  |
| --- |
| **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**

**6)**

|  |
| --- |
| **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):**

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

**return last2+last2+last2**

**7)**

|  |
| --- |
| **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:-1]**

**8) 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):**

**first=str[0]**

**second=str[1]**

**return str[2:]+first+second**

**9) 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+'>'**

**10) 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**

**elif len(str)==0:**

**return ''**

**else:**

**return str[0:2]**

**11) 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):**

**l1=len(a)**

**l2=len(b)**

**if(l1<l2):**

**return a+b+a**

**elif(l2<l1):**

**return b+a+b**

**LISTS**

**1)**

|  |
| --- |
| **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)**

**2)**

|  |
| --- |
| **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):**

**if a[0]==b[0] or a[-1]==b[-1]:**

**return True**

**return False**

**3)**

|  |
| --- |
| **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[-1],nums[-2],nums[-3]]**

**4) 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[-2],b[-2]]**

**5)**

|  |
| --- |
| **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 or len(nums)>1) and (nums[0]==nums[-1])**

**6) 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):**

**k=0**

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

**k=k+nums[i]**

**return k**

**7) 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[-1]:**

**return [nums[0],nums[0],nums[0]]**

**elif nums[0]<nums[-1]:**

**return [nums[-1],nums[-1],nums[-1]]**

**elif nums[0]==nums[-1]:**

**return [nums[0],nums[0],nums[0]]**

**8)**

|  |
| --- |
| **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[-1]]**

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

**make\_pi() → [3, 1, 4]**

**def make\_pi():**

**return [3,1,4]**

|  |
| --- |
| **10)**  **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[-2],nums[-1],nums[0]]**  **11)** |
| **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):**

**l=len(nums)**

**if l==0:**

**return 0**

**if l<2:**

**return nums[0]**

**else:**

**return nums[0]+nums[1]**

**12) 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 2 in nums or 3 in nums:**

**return True**

**else:**

**return False**