Beyond Blocks

Part II

From last time...

```
def hailstone(n):
    """Print the hailstone sequence starting at n and return its length.
    >>> a = hailstone(10)  # Seven elements are 10, 5, 16, 8, 4, 2, 1
    10
    5
    16
    8
    4
    2
    1
    >>> a
    7
    """
    "*** YOUR CODE HERE ***"
```

“You had me at Hello World!”

arrrr = “Hello World!”

```
>>>print(arrrr)
Hello World!
>>>“AA” + “AA”
‘AAAA’
>>>“AA” * 2
‘AAAA’
>>>arrrr[1: 5]
‘ello’
```

```
>>>arrrr
‘Hello World!’
```

How long is it? (you dirty mind!)

```
>>>print(len(“Hi”))
2
>>>print(arrrr.count(“r”))
1
>>>print(arrrr.index(“e”))
1
```

Exercise

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

def extra_end(str):

```
>>>extra_end('Hello')
'lololo'
>>>extra_end('ab')
'ababab'
>>>extra_end('Hi')
'HiHiHi'
```

Your Code Here

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).

def combo_string(a, b):

```
>>>combo_string('Hello', 'hi')
'hiHellohi'
>>>combo_string('hi', 'Hello')
'hiHellohi'
>>>combo_string('aaa', 'b')
'baaab'
```

Your Code Here

More in-depth with Strings

word = “Hello World!”

```
word[start:end]      # items start through end-1
word[start:]         # items start through the rest of the list
word[:end]           # items from the beginning through end-1
word[:]              # a copy of the whole string
```

Slicing and other tools

word = “Hello World!”

```
>>>print(word[:3])
Hel
>>>print(word[-3:])
ld!
>>>print(word[3:])
lo World!
>>>print(word[:-3])
Hello Wor
>>>print(word[100:101])
```

Random Generator

```
from random import randint

>>>randint(0, 10)
6
>>>randint(10, 100)
45

import random

>>>random.random()
(Any arbitrary decimal number)
```

Exercise

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

def cat_dog(str):
>>>cat_dog('catdog')
True
>>>cat_dog('catcat')
False
>>>cat_dog('1cat1cadodog')
True
***Your Code Here ***

Return a random number if and only if your PHRASE contains the word “tell me please”.

def tell_me(phrase):
>>>tell_me(“tell me”)
“You must learn to say ‘Please’”
>>>tell_me(“tell me please”)
40
>>>tell_me(“please”)
“What do you want me to tell you?”
>>>tell_me(“kick me please”)
“No.”
***Your Code Here ***

Replbce!

intro = “Hello, My name is Song”

```
>>>intro.replace(“Song”, “Shabolabadadingdong”)
'Hello, My name is Shabolabadadingdong'
>>>“replace”.replace(“a”, “b”)
‘replbce’
```

More Strings!!!

```
>>>“x” in “xyz”
True
>>>“xyz” in “x”
False
>>>word = “hi, my name”
>>>word.split(“,”)
['hi', ' my name']
>>>”ABC”.lower()
‘abc’
>>>”abc”.upper()
‘ABC’
```

```
>>>for word in “Hello”:
        print(word)

H
e
l
l
o
```

Exercise

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

def count_code(word):

```
>>>count_code('aaacodebbb')
1
>>>count_code('codexxcode')
2
>>>count_code('cozexxcope')
2
```

***Your Code Here ***

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

def xyz_there(word):

```
>>>xyz_there('abcxyz')
True
>>>xyz_there('abc.xyz')
False
>>>xyz_there('xyz.abc')
True
```

***Your Code Here ***

Lists

DON'T EVER NAME YOUR LIST "list", BAD PRACTICE!
lst = [1, 2, 3, 4, 5]

List is exactly like a String in terms of slicing and getting an item at an index.
So,

```
>>>lst[0]
1
>>>lst[1:3]
[2, 3]
```

There're also other methods you can use for list. For best, you can read the documentations for list.

But I'll save you some work and show you some of the most common ones.

```
>>>lst.append(5)
>>>lst
[1, 2, 3, 4, 5, 5]
>>>lst.pop()
5
>>>lst
[1, 2, 3, 4, 5]
```

```
>>>lst.remove(2)
>>>lst
[1, 3, 4, 5]
>>>lst.index(3)
1
>>>lst.append(1)
>>>lst.count(1)
2
```

Fingers Exercise

Return a boolean where true means that LST1 is equal to LST2 and false otherwise.

def equals(lst1, lst2):
>>>equals([1, 2, 3], [1, 2, 3])
True
>>>equals([2, 5], [4, 2, 22])
False

***Your Code Here ***

Return a number where number is the amount of counts there are based on given input KEY in LST1.

def count(lst1, key):
>>>count([2, 2, 0], “even”)
3
>>>count([1, 2, 3, 4], “odd”)
2

***Your Code Here ***

More Lists

```
>>>mylist = [2, 6, 4, 1]
>>>mylist.sort()
>>>mylist
[1, 2, 4, 6]

>>>mylist[1:3]       #this will create a new list
>>> mylist = mylist + mylist[1:3]
>>> mylist
[1, 2, 4, 6, 2, 4]

>>>mylist * 2
[1, 2, 4, 6, 2, 4, 1, 2, 4, 6, 2, 4]
```

```
>>>mylist[1] = 5
>>>mylist
[1, 5, 4, 6, 2, 4]

>>> mylist[3:6] = [1, 5, 7, 3, 5]
>>> mylist
[1, 5, 4, 1, 5, 7, 3, 5]
```

More Exercise

Return a number of the maximum duplicates there are in the list LST.

def countdups(lst):

```
>>>lst = [1, 2, 2 ,2, 3, 4]
>>>countdups(lst)
3
```

***Your Code Here ***

Mutate the list LST such that LST is reversed. DO NOT return a new list.

def reverse(lst):

```
>>>lst1 = [1, 2, 3, 4]
>>>reverse(lst1)
>>>lst1
[4, 3, 2, 1]
>>>lst2 = [3, 4, 5]
>>>reverse(lst2)
>>>lst2
[5, 4 ,3]
```

***Your Code Here ***

Dictionary

Just like a dictionary book, a dictionary in Python contains the ***“key”*** which is the ***“word”***, and the ***“value”*** which is the ***“definition”***.

```
>>>phone = {‘jack’ : 2419051481, ‘jamie’ : “1800gotohell”}
>>>phone[‘jack’]
2419051481
>>>phone[‘jamie’]
“1800gotohell”
```

More Dictionary methods

```
>>>list(phone.keys())
[‘jack’, ‘jamie’]

>>>del phone[‘jack’]
>>>phone
{‘jamie’ : “1800gotohell”}
```

Dictionary is an iterable, but you can ***only*** iterate over a dictionary’s ***keys***, not its ***values***.

Higher-Order Functions!!!

In Snap:

⇒ map(function, iterable)

⇒ filter(function(boolean), iterable)

⇒ reduce(function, iterable)*

In Python:

HOFs Challenges

Return a list of all the lowercase letters in an arbitrarily large string STR of mostly uppercase letter.

def cipher(str):
>>>cipher('HHaHHbHHcH')
[a, b, c]
>>>cat_dog('aJKLMNOPb')
[a, b]

***Your Code Here ***

Get the cipher string from BeyondBlocks.github.io